mle

A Programming Language for Building Likelihood Models

Version 2

Darryl J. Holman

Table of Contents

mle

A Programming Language for Building Likelihood Models

Version 2

Darryl J. Holman

Department of Anthropology
Center for Studies in Demography and Ecology
Center for Statistics and the Social Sciences
The University of Washington
Box 353100
Seattle, WA 98195

holman@pop.psu.edu

Suggested citation: Holman, Darryl J. (2000) *mle*: A Programming Language for Building Likelihood Models. Version 2. http://faculty.washington.edu/~holman/mle.

mle** List:* There is an email list for ***mle users to receive update and bug notices. To subscribe, send an email message to majordomo@pop.psu.edu with the text "subscribe mle" as the body of the email message.

TABLE OF CONTENTS

Table of Contents

Table of Contents

INTRODUCTION

mle is a simple programming language for building and estimating parameters of likelihood models. The language was originally intended for building and estimating the parameters of survival models, but the language has evolved to be general enough to estimate parameters for many other types of likelihood models. Indeed, the language attempts to be a general purpose tool for likelihood estimation.

This chapter provides an overview of ***mle***. The basic concepts of the programming language are introduced and some examples are given. Details, and more formal descriptions of the ***mle*** programming language, are saved for later chapters. Examples of ***mle*** programs and program fragments are sprinkled throughout this manual. A later chapter is devoted to examples of different type of likelihood models.

This manual gives only a superficial treatment of topics like probability theory, probability models, and maximum likelihood methods. In order to write ***mle*** programs, you will need a basic understanding of these topics. Some helpful, generally applied, introductions to maximum likelihood estimation can be found in Edwards (1972), Hilborn and Mangel (1997), Holman and Jones (1998), Nelson (1982), Pickles (1985), Wood et al. (1992).

Programs written in ***mle*** are, in many respects, similar to those written in SAS, S+, SPSS, BMDP, or many other statistical programming languages. The language consists of keywords like `MODEL`, `END`, `DATA`, and so on. Like all languages, ***mle*** has rules of syntax that must be strictly followed to produce a valid program. The resulting ***mle*** program is translated into actions (like parameter estimation) by the ***mle*** interpreter.[1]

The ***mle*** interpreter typically works with three files: the ***mle*** program file, the data file, and the output file.

The program file. This is the program that you have written in the ***mle*** programming language. The first line of this file begins with the word `MLE` and the program ends with a matching `END`. In between these two comes the program, which includes naming the data file and the output file, describing how to read in and transform data into a series of observations, and specifications of one or more likelihood models along with parameters to find. Parameter estimates are then found by an iterative search that maximizes the likelihood given a set of observations. The resulting parameter estimates are then written to an output file.

The ***mle*** program is created as an ordinary text file using almost any editor. You can create and edit the ***mle*** program using the *EDIT* command (in DOS), *vi*, *pico*, or *emacs* (in Unix), or any other editor that will read and write a file as ASCII text. Word processors, such as Microsoft *Word*, can be used as well, but you must remember to save your work using the "*text (with line breaks)*" option.

The data file. This file contains lines of observations. The observations are read, and perhaps transformed, when the ***mle*** program is run. The observations are then used with the likelihood function (specified in the ***mle*** program file) to find parameter estimates. Data files are standard ASCII text files. Typically, one line in the file represents one observation (although a single observation can span more than one line, and one line can represent multiple identical observations).

[1] Notice that ***mle*** has two distinct meanings in this document. First, it is a programming language for building likelihoods. Second, it is the name of the computer program that interprets the language and finds maximum likelihood estimates of model parameters.

```
MLE
  TITLE = "Distribution of gestational age" {Data are from Hammes
                                             and Treloar(1970) Am J Pub
                                             Health 60:1496-1505}
  MAXITER = 50             {Maximum number of iterations allowed}
  EPSILON = 0.0000001      {Criterion for convergence of the model}
  DATAFILE("hammes.dat")   {Opens the input data file}
  OUTFILE("hammes.out")    {Opens the output file}

  DATA
       {Data are interval censored and are
        in units of days as per Table 2 of Hammes and Treloar}
    topen     FIELD 1            {time at opening the interval}
    tclose    FIELD 2            {time at closing the interval}
    frequency FIELD 3            {Frequency from Menstrual History Program}
  END {data}

  MODEL
    DATA                        {function to loop through all observations}
      PDF NORMAL(topen, tclose) {Define the parametric distribution}
        PARAM    mean      LOW = 100  HIGH = 400 START = 270 END
        PARAM    stdev     LOW = 0.1  HIGH = 100 START = 20  END
      END {pdf}
    END {data}
  RUN
    FULL    {run the model with both parameters free}
  END {model}

END  {of program}
```

Figure 1. Program to estimate parameters for the distribution of gestational ages at birth.

Within each observation is a series of fields that are separated by spaces, tabs, commas, or some other user-specified delimiter. One or more of these fields are read into variables.

The output file. This is where results are written. The name of the output file is specified in the ***mle*** program file. The program file also specifies what kind of result will be written to the output file, and how much of the details will be included.

You can also specify that ***mle*** write partial results and messages to the screen (or standard output as it is called). This is helpful for monitoring progress while estimation is taking place.

An Example

Figure 1 is a simple ***mle*** program that illustrates the most important parts of the ***mle*** programming language. The problem at hand is to estimate the distribution of gestational ages at birth given the observations shown in Figure 2. These observations are counts of gestational ages at birth that were recorded at a resolution of one week for all but the first and last rows of observations. We will use survival analysis to estimate the parameters (μ and σ) of the normal distribution that best describes these data.

This is an example of survival analysis with interval censored observations. In this example, observations are given as frequencies within each interval; that is, each line in the data file represents many observations.

Structure. There are four basic constructs in a typical *mle* program. They are assignment statements, procedure statements, the DATA statement and one or more MODEL statements. Assignment statements allow you to define and change program variables, some that affect the behavior of the DATA and MODEL statements. The DATA statement describes the format of the input data file, and provides simple data transformations and mechanisms to drop observations. The MODEL statement defines the likelihood function along with the parameters to be estimated. A second part of each MODEL statement is the RUN statement that specifies how the model is to be estimated.

Program constants and variables. A number of variables (e.g. MAXITER) are pre-defined in *mle*. Frequently, you will want to change the value of these variables in order to fine tune the behavior of the program, how the language is interpreted, the type of output produced, etc. In the example program, the value of MAXITER is changed from the default value of 100 to a maximum of 50. MAXITER is the maximum number of iterations allowed for finding the parameters. Notice the TITLE variable is also assigned to a string variable (i.e. a series of characters). The TITLE variable is simply written to the output file. The variable EPSILON is assigned a value as well. This variable determines how precisely the parameters are to be found: normal convergence occurs when the change in the log-likelihood from one iteration to the next falls below this value.

Comments. These can be placed throughout the body of a program by enclosing the text in curly brackets, { and }. Likewise, the curly brackets can be used to effectively remove large sections of code. A second way to comment out all or part of a single line is to put a pound sign # at the point where you want the comment to begin. *mle* ignores all text that follows the pound-sign through the end of the line.

Reading data. The data file specified in Figure 1, called hammes.dat, is shown in Figure 2. Data files are standard ASCII text files of numbers. The numbers are organized into a series of fields. Each filed is usually delimited by white space (tabs or spaces as used in Figure 2) or commas. You can specify your own list of delimiters by changing the value of the variable called DELIMITERS (see the DATA chapter for details).

The data in Figure 2 are structured as three columns of numbers. The first field is the last observed gestational age prior to birth. The second field is the observed gestational age after a birth was observed. These two times form an interval within which the birth occurred (i.e. the birth occurred at some unknown time within this interval). The third field is the number of births that were observed within the interval.

The way in which the data file is read is specified by the DATA statement in Figure 1. The three variables, TOPEN, TCLOSE, and FREQUENCY, that come between DATA and its matching END, are read in for each observation (i.e. each line in Figure 2). In fact, each of these variables will be created as an array, each having twenty elements. Each element corresponds to a line read in from the data file.

The variable called frequency is a special name because *mle* will treat variables with the name frequency (and

```
0   141 0
141 196 9
197 217 11
218 224 2
225 231 12
232 238 17
239 245 22
246 252 40
253 259 69
260 266 134
267 273 324
274 280 653
281 287 724
288 294 382
295 301 125
302 308 47
309 315 26
316 322 10
323 329 1
329 -1  6
```

Figure 2. Data file read by the program in Figure 1. The first two columns define an interval within which a birth occurred. Note the last row has -1 to denote an opened (right censored) observation. The third column is the number of pregnancies that terminated within each interval.

freq as well) as a count of repeated observations. Thus, the contribution to the likelihood from each observation will be the same as if the number of observations specified by frequency had been read in from the file.

Likelihood model. The next part of the program is the MODEL statement. The MODEL statement consists of two parts: an expression that comes between the MODEL and RUN part that defines the likelihood, and a list of one or more run specifications that come between the RUN and END part of the statement.

Within the MODEL...RUN part of the statement is a single function that defines the likelihood. In this example, we are specifying the likelihood:

$$L = \prod_{i=1}^{N} \left[S(t_{open_i} | \mu, \sigma) - S(t_{close_i} | \mu, \sigma) \right]^{frequency_i} \quad (1)$$

where N is the number of age categories (i.e. the number of lines of observations), *frequency* is the frequency of observations per age category, S() is a survival density function for the normal distribution, t_{open} and t_{close} are the two times read from the data file into the variables topen and tclose, and μ and σ are the parameters that will be explored to maximize the likelihood.

Notice that the first part of the likelihood expression is a DATA...END function. This function specifies that observations are to be "fed" to the likelihood one at a time, corresponding to the product ($\prod$) shown in the likelihood above. It is very important that you do not confuse the DATA *function*, found within the MODEL statement, with the DATA *statement* discussed above. The DATA *function* loops through all observations that were previously read in by the DATA *statement*. Within the DATA...END function comes the rest of the likelihood, which is shown to the right of the $\prod$ in likelihood (1) given above. The likelihood within the data function is called for each observation in turn. The resulting individual likelihood is computed, the log of that likelihood is taken, multiplied by the FREQUENCY for the current observation, and added to the total likelihood. In other words, the DATA...END function returns the *total log-loglikelihood*, given a series of *observations* and an expression for an *individual likelihood*.

Nested within the DATA function is the PDF function, which makes up the parametric model for the likelihood function. A NORMAL distribution is specified, and we pass it two arguments (topen, tclose). These two arguments (which were read from column 1 and 2 of the data file) different from each other, so the PDF function returns the area under a normal PDF between the two points. The area corresponds to the probability of observing an birth within the interval. If we had only specified a single argument or if both arguments had been equal to each other, the PDF function would have returned the probability density at that point. Within the PDF NORMAL function call are two PARAM functions. These functions define parameters that will be changed in order to maximize the likelihood. Naturally, you are able to specify limits, starting values, etc. for these parameters.

Between the RUN and the END part of a MODEL statement comes a list specifying how to run the model. The full model is run by specifying FULL. Various reduced forms of the model can be run by specifying a REDUCE command. More details on this are given below and in a later chapter.

```
Distribution of gestational age
Parameter file: hammes.mle
Input data file name: hammes.dat
Output file name: hammes.out
    3 variables read.

18 lines read from file hammes.dat
18 Observations kept and 0 observations dropped for each variable.

ROW         topen      tclose   frequency
MEAN   258.722222  253.555556  144.111111
VAR    5338.56536  6032.37908  51267.3987
STDEV  73.0654868  77.6683918  226.423052
MIN    0.00000000  -1.0000000  0.00000000
MAX    329.000000  329.000000  724.000000
New model:  Distribution of gestational age

METHOD = DIRECT
Maximum Iterations MAXITER = 50
Convergence at EPSILON = 0.0000001000

Results with estimated standard errors:
  Log Likelihood = -5915.1352 after 4 iterations.  Delta(LL)=0.00000000
PDF NORMAL with 2 free parameters
      Name Form       Estimate         Std Error          t        against
      mean          279.7654969512   0.267153495349    1047.20882123   0.0
     stdev          13.04605798312   0.126880990969    102.821217611   0.0

Variance/covariance matrix:
0.07137099008  0.00860300945
0.00860300945  0.01609878587

Likelihood CI Results:
  Log Likelihood = -5915.1352 after 4 iterations.  Delta(LL)=0.00000000
PDF NORMAL with 2 free parameters
        Name Form       Estimate          Lower CI          Upper CI
        mean          279.7654969512   279.1863052702    280.3447034638
       stdev          13.04605798312   12.64289497881    13.47052893809
```

Figure 3. Output generated by the program in Figure 1.

Running. The *mle* program is run by typing the line `mle hammes.mle` at the command line prompt (see Chapter 2 for details). The output that results from the example program and data file is given in Figure 3. The first section of the output provides summary statistics for each of the variables read from the input data file. The parameter estimates are given in two ways: once with estimated standard errors, and once with likelihood confidence intervals. The standard error report also shows a *t*-test of the hypothesis that the mean parameter estimate is zero.

Basic Outline of an mle Program

The easiest way to write an *mle* program is to begin with a working program like that given in Figure 1, or at least an outline like that given in Figure 4. Every program begins with the word `MLE` and ends with the matching word `END`.[2] Any text

[2] Throughout this manual, *mle* programs are shown with indentation to show, for example, the matching `MODEL` and `END`. This so-called pretty-printing is not necessary; *mle* is a free-format language. Nevertheless, the use of indentation and copious commenting will greatly aid in proper program development and debugging. This manual uses two spaces to indent each natural "level". Key words that are a part of *mle* are always upper-case letters and user-defined words are lower-case (again, this is not required since *mle* is not case sensitive). Finally, a matching END is usually followed by a comment denoting what key-word the end matches. This last convention is particularly useful for complex programs that involve many nested functions.

after the final END is ignored. Between the MLE and its matching END comes the body of an *mle* program. Four types of statements occur within the body: the DATA statement, one or more MODEL statements, procedure calls, and assignment statements. Each type of statement is briefly discussed below.

Assignment Statement

Assignment statements may be placed anywhere within the body of the *mle* program—that is, within the MLE and its matching END.[3] A great number of pre-defined variables are available in *mle* that change or fine-tune the behavior of the program. Assignment statements are used to change the value of these variables. Some brief examples are:

```
MAXITER = 100          {Set the maximum number of iterations}
EPSILON = 0.0000001    {Set the criterion for convergence}
PRINT_OBS = TRUE       {prints all observations after transformations}
```

The assignment statement is generically defined as <*variable name*> = <*expression*>. The <*variable name*> name can be a preexisting variable (e.g. MAXITER, EPSILON), or is a user-defined variable.

The <*expression*> that follows the equal sign can be a simple constant, another variable, or a complex mathematical expression. The details of the syntax and the many functions that can be used to make up expressions are given in a later chapter. The following are some example of assignment statements using expressions:

```
pie      = PI
bmi_max  = weight_max/height_max^2
total    = e1_count + e2_count + e3_count + e4_count
last_sge = IF linear THEN max_age ELSE SQRT(max_age) END
area     = PDF NORMAL(-2, 2) 1, 3 END        {gives area
                          from -2 to 2 for N(1, 3)}
one      = SIN(total)^2 + COS(total)^2
```

These are all examples of assignments that return *real* number results. There are, in fact, five different *types* supported by *mle*: *real*, *integer*, *boolean*, *string*, and *character*. Variables can be defined for each type. Additionally, expressions can be created for each of these types.

```
MLE
  DATAFILE("...")
  OUTFILE("...")
  TITLE = "...
  MAXITER = 100


  DATA
    <Data specification>
  END

  MODEL
    <Expression>
  RUN
    <Run specification>
  END
 <Additional MODELs>
END
```

Figure 4. The structure of an *mle* program

Real. Variables of this type represent the continuous real number line.[4] Many mathematical functions like SIN(), EXP(), and BESSELI() return real values, and so the variable to which these functions are assigned must be type REAL as well. Real variables can always take on integer values, but integer variables must use the ROUND() or TRUNC() functions to convert a real number to an integer value.

Integer. Variables of this type can only take on whole number values over a machine-dependent range of numbers. For example, on DOS computers this range is [-2,147,483,648, 2,147,483,647]. Arguments of some functions *require*

[3] Normally assignment statements do not occur within the DATA...END and MODEL...END statements. Assignment-like statements occur within the DATA statement for transformations. Additionally, the PREASSIGN and POSTASSIGN functions allow a list of one or more assignment (or other) statements to be used. Finally, within the MODEL statement, there are several other uses for assignment-like statements, like to define start, highest, and lowest values of parameters.

[4] Be aware, however, that the computer representation for real numbers is not strictly continuous. Occasionally this leads to difficulties with round-off errors.

INTEGER type variables. For example, the INC(x) function requires that x is an integer type. A number of functions (ROUND, TRUNC) can convert real types into integer types.

Boolean. Variables of this type take on one of two states: TRUE or FALSE. No other value is allowed or recognized. Boolean expressions are frequently used to test conditions. For example, the IF...THEN...ELSE...END function evaluates the first expression (between the IF and THEN) to either TRUE or FALSE and decides which of the remaining two expressions will be evaluated and returned.

String. Variables of this type hold a sequence of character constants. When written as a constant in a program, these constants consist of a sequence of characters, enclosed within double-quotes ("). String variables are typically used to assign file names, titles, etc. A few functions take on string (or character) variables and return a string. For example, the CONCAT(s1, s2) function will add together two string variables and return it as a longer string.

Character. Variables of this type take on the value of a single character. When written as a constant in a program, character constants consist of a single character enclosed within single quotes ('). Character constants are not typically used within a user's program, but are available if needed. Usually, character constants and variables can be used anywhere a string variable is allowed.

When a variable is first used in an assignment statement, its type is determined based on the type returned from the expression on the right-hand side of the assignment. Here are some examples to illustrate the point:

```
large_data = N_OBS > 5000                    {large_data will be type BOOLEAN}
subtitle   = "Analysis: " + DEFAULTOUTNAME   {subtitle will be type STRING}
nine       = 3 * 3.0                         {nine will be REAL}
five       = 2 + 3                           {five will be INTEGER}
```

You can explicitly define the *type* for a variable when it is first referenced in an assignment statement. Here are some examples:

```
c:STRING  = 'x'      {c would default to CHAR, but will be a STRING variable}
nine:REAL = 3 * 3    {nine would default to INTEGER, but will be a REAL variable}
t:BOOLEAN = TRUE     {t is explicitly declared as boolean, although this is the default}
ang:REAL = SIN(2*pi) {ang is explicitly declared as real, although this is the default}
```

Multidimensional arrays and matrices of all *types* are supported by ***mfe***. Arrayed variables must be explicitly defined the first time the variable is mentioned in the program. The format is <*var*> : <*type*> [*min1* TO *max1, min2* TO *max2*, . . .]. Some examples of declarations are:

```
s : STRING[1 TO 5]                      {Defines a one-dimensional array of strings}
r : REAL[1 TO 10, 1 TO 10]              {Defines a 10 x 10 matrix}
b : BOOLEAN[0 TO 1, 0 TO 1, 0 TO 1]     {Defines a 3 dimensional BOOLEAN array}
```

An entire array can be initialized to a single value in an assignment statements. Examples are:

```
s : STRING[1 TO 5] = ""        {Defines s and initializes all values to an empty string}
r : REAL[1 TO 10, 1 TO 10] = 0 {Defines a 10 x 10 matrix and initializes everything to 0}
```

Arrayed variables are accessed by using brackets to denote subscrips. The following example creates an array of radian angles for integral degree angles, and prints out a table of sine values:

```
r : REAL[0 TO 359]
FOR i = 0 TO 359 DO
  r[i] = DTOR(i)
  writeln("Sin(" i ") = " SIN(r[i]) )
END
```

Data Statement

A single DATA...END statement is usually used in a program. The purpose of the statement is to read in an input file and transform the data in the file into a set of observations. The DATA statement defines the format of the data file and provides declarations for each variable that is to be read in from the file. Variables declared within a DATA statement are a special array of real number. After the matching END is read, the DATA statement immediately reads in the file and performs any of the transformations specified. Details on the DATA...END statement are given in a later chapter.

The ultimate purpose of a DATA *statement* is to create a series of observations. The observations will typically be used to compute likelihoods. Within a MODEL statement, you can use the DATA *function* to evaluate the likelihood, one observation at a time. Do not be confused by the fact that there is both a DATA statement and a DATA function. They complement each other. Simply remember that a DATA statement is used as a statement, and there is typically one such statement per ***mle*** program. The DATA function can only be used as part of an expression—typically only within the likelihood expression of a MODEL statement. Each MODEL statement will usually have one DATA function included as part of the likelihood specification. The DATA function corresponds to the product ($\prod$) over all observations in the likelihood [e.g.(1)] or the summation overall all observations (Σ) in a log-likelihood.

Model Statement

The MODEL...RUN...END statement defines the underlying probability model used by ***mle*** and also defines constraints on which parameters are to be estimated. Between MODEL and RUN is a single expression that is the likelihood. Within the likelihood is one or more PARAM...END functions. These define the parameters, whose values will be found to so that the likelihood is maximized. One of the most important aspects of learning ***mle*** is the design and construction of the expression for the likelihood. A later chapter gives a number of examples for different types of likelihoods.

A list of *run specifications* is given between the RUN and the END part of the MODEL statement, this provides a way of evaluating the full model as well as a series of nested or reduced models. If all of the parameters (defined by PARAM...END functions) are to be found, a simple FULL command is placed between the RUN and its matching END. Reduced models, where one or more parameters are constrained to a constant or another parameter, are specified as REDUCE followed with a list of one or more "reductions". For example, you might constrain a parameter called mean to be zero and only allow the parameter called stdev to be found. Then you would put REDUCE mean = 0 between the RUN and the END. Any number of REDUCE commands (along with one FULL) can be used in a single model. The various forms of the model will be evaluated in turn. Additional details are given in a later chapter.

Procedure Statements

mle has a number of pre-defined intrinsic *procedure statements*. These are pre-defined procedures that perform some specific task. Unlike a function call, procedures do not return some value as part of an expression. Hence, procedures are called in the same context as an assignment statement. Here are some common examples of procedure statements:

```
DATAFILE("hammes.dat")                {Tells mle to use the file hammes.dat as the data file}
OUTFILE("hammes.out")                 {Defines the output file for mle to use}
SEED(9734)                            {Seeds the random number generator}
```

Other Statements

mle provides a number of standard programming statements. The statements should be familiar to anyone with computer programming experience in BASIC, FORTRAN, C, or Pascal. You will not ordinarily need to use these statements, but they are provided in the event you do need them. The statements are illustrated below without any details. The full details are given in a later chapter:

```
FOR x = 1 TO 10 DO
  y = x^2
  PRINTLN("x is ", x, " and x^2 is ", y)    {PRINTLN() writes to the log file}
END {for}

i = 2
WHILE i < 1000 DO
  i = i^2
  WRITELN(i)
END {while}

SEED(234)
REPEAT
  PRINTLN("Still working")
  IF RAND <= 0.2 THEN
    DONE = TRUE
  ELSEIF IRAND(1, 100) = 100 THEN
    DONE = TRUE
  ELSE
    DONE = FALSE
  END {if}
UNTIL done
```

A Note About Parameters

The ultimate goal of putting together a likelihood model is to estimate one or more *parameters* of the model. The `PARAM...END` function (described later) provides a method to define the parameters that are to be estimated. This use of the word "parameter" can be confusing, so lets clear it up right from the start. In any mathematical language, we can refer to a function's arguments as "parameters". For example, in the statement $a = \sin(b)$, sin() is a function with one "parameter". This manual will avoid the word "parameter" in this general sense. Instead, the word *argument* will be used to refer to the arguments of a function in a general sense. So, the sin() function has one *argument*.

As used in this manual, the word *parameter* in ***mle*** refers to an unknown quantity of a probability model whose value is to be estimated. Parameters, in this sense, are frequently arguments to functions, but not all arguments are parameters. Parameters are sometimes the constants defined within a function. For example, in the well-known equation for the slope of a line, $y = mx + b$, we would call m and b the parameters of the equation, and x the argument. This is clearer when we rewrite the equation for a slope as $f(x \mid m, b) = mx + b$, which is read, "f of x given m and b. . . ." This function has a single argument x, and the parameters are m and b. Typically a series of x values are known, and the goal is to find the best values for parameters m, and b. By "best", of course, we mean the best in some statistical sense. In ***mle***, m and b would be called parameters if and only if they were quantities to be estimated.

The one exception to the usage of the word parameter is for the built-in probability density functions in ***mle*** , where we refer to the *intrinsic parameters*. For example, the normal distribution, $f(t|\mu, \sigma)$, has two intrinsic parameters, μ and σ. Typically we wish to estimate these intrinsic parameters. If so, the intrinsic parameters μ and σ are also parameters.

As described later, most probability density functions take four arguments for the argument *t* so that the cumulative density, survival density, area under the probability density, probability density, hazard function, and all of the above with right and left truncation may be specified. Thus, in the syntax of ***mle*** , there is a natural delineation between arguments and intrinsic parameters. Consider the following function call: `PDF NORMAL(0, 4, 0, 40) 10, 20 END`. This function call has the four "time" arguments 0, 4, 0, and 40, which specifies a normal distribution truncated over the range 0 and 40, and the area between 0 and 4 is returned. The two intrinsic parameters of the normal are passed as $\mu = 10$ and $\sigma = 20$. There are no "parameters" in this example, simply because there is no `PARAM` function specified.

Differences between Version 1 and Version 2

Changes and new features in version 2

There are a number of syntax differences and other changes between ***mle*** version 1 and version 2. Here is a summary of the most important changes:

- General algebraic expressions are now recognized. Standard operators include: `+`, `-`, `*`, `/`, `^`, `and`, `or`, `xor`, `not`, `mod`, `div`, `shl`, `shr`, `>`, `<`, `<>`, `=`, `==`, `>=`, `<=`. These operators can be used to build algebraic and boolean expressions of nearly unlimited complexity. Both = and == are allowed for specifying boolean relationships. The standard operator precedence, common to most programming languages, is recognized by ***mle***:

Operator(s)	Precedence	Category
`- + not`	high	Uniary operators
`^`		Exponent operator
`* / div mod and shl shr`		Multiplying operators
`+ - or xor`		Adding operators
`= (or ==)  <> < > <= >=`	low	Relational operators

  The expression `-23+4*-2^3` is equivalent to `ADD(NEGATE(23), MULTIPLY(4, POWER(NEGATE(2), 3)))` which returns `-55`. Parenthesis can be used to override operator precedence. For example, 2*5 + 3*7 will evaluate each multiplication before the addition. Addition can be forced to occur first with parenthesis as in 2*(5 + 3)*7.

- The `DATA` statement has been rewritten to have a more intuitive transformation mechanism. The transformation looks like an assignment statement following the `FIELD` and `LINE` specification (if any). A list of `DROPIF` <expr> and `KEEPIF` <expr> statements can then be specified (replacing the old `DROP` and `KEEP` statements). Here are some examples:

```
DATA
  age     FIELD 1 = age*365.25 + 270  {convert to days since conception}
  weight  FIELD 2 = weight * 1000  DROPIF weight <= 0
  height  FIELD 3   KEEPIF height > 0
  bmi             = height/weight^2
END {data}
```

The formal specification for each variable is this

<var> [FIELD x [LINE y]] [= <expr>] [DROPIF <expr> | KEEPIF <expr> ...]

The first example above reads a value in the first field of the data file and assigns the value to the variable age. After that, the expression age*365.25 + 270 is evaluated and the result assigned to the variable age. The second example reads the second field and assigns the value to the variable weight. Following that, the expression weight*1000 is evaluated and assigned to the variable weight. Then the expression weight <= 0 is evaluated. If TRUE, the observation is dropped. If not, the observation is kept.

- Observations can now be simulated or otherwise created within ***mle***, without reference to a data file. This is done by setting CREATE_OBS to some positive value. The following example will create 100 uniform random observations:

```
CREATE_OBS = 100
DATA
  v1      FIELD 1 = RAND
END {data}
```

- A number of useless functions that were used with the old data transformations have been eliminated, e.g.: ONE, SECOND, ONEIF, RESPONSE, etc.

- A number of new functions have been added, e.g.: DEFAULTOUTNAME, FISHER, ISODD, STRING2REAL, INT2STR. A fairly complete set of functions are now available to work with calendar dates. A full list of simple functions can be generated by typing mle -h functions.

- The PREASSIGN and POSTASSIGN functions have been generalized so that any single statement is allowed in the statement part of the function. By using a BEGIN ... END block, more than one statement can be used in the assignment part of the functions. For example:

```
PREASSIGN
  BEGIN  {This is the statement part}
    r : REAL[0 TO 359]
    FOR i = 0 TO 359 DO
      r[i] = DTOR(i)
    END {for}
  END {begin – this is the end of the statement part of the PREASSIGN}
  PDF NORMAL(a, b) c, d END  {This is the function returned by PREASSIGN}
END {preassign}
```

- The conditional expressions in the IF THEN ELSE END and LEVEL functions take a boolean expression of any complexity, e.g., IF (a = b) AND (c^2 + 2 <= 23) OR (d > 1) THEN ... ELSE ... END.

- The IF...THEN...ELSE...END function has been generalized so that multiple ELSEIF...THEN... conditions may be added. The following assignment is an example:

```
status = IF height < 48 THEN
           -1
         ELSEIF (height >= 48) and (height <= 60) THEN
           0
         ELSE
           1
         END {if}
```

- Types can be optionally defined for variables when they are first encountered. Valid types are INTEGER, REAL, CHAR, STRING, BOOLEAN. For example:

```
x : REAL = 23   {x would be integer, but is defined to be real}
c : STRING = '!'   {c would be char, but is defined to be string}
```

- In general, types are handled better. Adding two integers together, for example, returns an integer. The `IF...THEN...ELSE...END` function can return any type, but the type after the `THEN` must match the type after the `ELSE`.

- Multidimensional arrays are supported for all types. Subscripted values are accessed as, for example, `z[i, j, k]`. Arrays are declared as

```
a : REAL[1 TO 5, -1 TO 1]  = 0   {Declare and initialize matrix a}
```

- A new `DERIVATIVE` function numerically finds the value of a derivative at a specified point along some function. For example, `DERIVATIVE x = 2, 3*x^2 + 2*x + 4 END`, which is the derivative of $3x^2 + 2x + 4$ evaluated at $x = 2$, returns 14.0.

- The new `FINDMIN` function finds the value that minimizes a bounded function. An example is `FINDMIN x (0, 2*PI) COS(x) END`, which finds a minimum of the function cosine(x) between 0 and 2π. It returns 3.1415925395570 (π is an exact solution). The accuracy of the solution may be specified as a third argument within the parenthesis.

- The new `FINDZERO` function finds the value of an argument for which the function goes to zero. An example is `FINDZERO x (0, PI) COS(x) END`, which finds a value of x for which cosine(x) is zero. It returns 1.5707963267949 (which is close to the exact solution of $\pi/2$). The accuracy of the solution may be specified.

- An important syntactical change is that every `PARAM` function must have a matching `END`.

- The default `FORM` for the `PARAM` function is `NUMBER` if no covariates are specified and `LOGLIN` if one or more covariates are specified.

- The `COVAR` specification part of the `PARAM` function has been generalized to `COVAR` <expr> <expr>. A typical specification is

```
PARAM x  LOW=0  HIGH=100 START=25
  COVAR z PARAM beta_z LOW=-5 HIGH=5 START=0 END
END
```

  Nevertheless, other expressions are legal. For example

```
PARAM x LOW=0 HIGH=100 START=25
  COVAR z 1
END {param}
```

- The PARAM options `HIGH`, `LOW`, `START`, and `TEST` are treated like assignment statements which are evaluated just prior to maximization. The right-hand side of the assignment can be any valid expression. For example,

```
PARAM a LOW = IF y > 3 THEN 0 ELSE 3  HIGH = x^2 + 2x - 4  START = y - 1 END
```

- The `CONST` part of the `MODEL` statement is longer supported.

- A number of procedures have been added that can be used wherever a statement is allowed, including

```
WRITE()      {writes to standard output}
WRITELN()    {writes a line to the standard output}
PRINT()      {writes to the output file}
PRINTLN()    {writes a line to the output file}
SEED()       {seeds the random number generator}
DATAFILE()   {defines the data file}
OUTFILE()    {defines the output file}
HALT         {halts the program}
```

- A variety of statements have been added that can be used wherever a statement is allowed, including

```
IF <bexpr> THEN <statements> ELSEIF . . . ELSE <statements> END
FOR <v> = <expr> TO <expr> DO <statements> END
BEGIN <statements> END
WHILE <bexpr> DO <statements> END
REPEAT <statements> UNTIL <bexpr>
```

Converting version 1 programs to version 2

Programs written in earlier versions of ***mle*** can be converted into later versions without much difficulty. The most important things to change are given below.

- Change all `INFILE = "mydata.dat"` statements to `DATAFILE("mydata.dat")` procedure calls.
- Change all `OUTFILE = "results.out"` statements to `OUTFILE("results.dat")` procedure calls.
- Change all `SEED = 5352` statements to `SEED(5352)` procedure calls.
- Eliminate all `CONST` blocks that may have been used at the beginning of `MODEL` statements. Instead, define the constant outside of the `MODEL` statement. Alternatively, use a `PREASSIGN` function within the `MODEL` statement to create temporary variables within that statement.
- Add an `END` after all `PARAM` functions.
- Some older versions of ***mle*** did not have or allow the `DATA...END` function within the `MODEL` statement. In more recent versions, a `DATA...END` function is almost always required to cycle through all observations in the data set. `MODEL` statements should usually look like this:

```
MODEL
  DATA
        {the rest of the likelihood goes here}
  END   {data}
RUN
  FULL
END     {model}
```

- Some older versions of ***mle*** used the keyword `FREQ` followed by a variable name *within* a `PDF` function to denote the a frequency variable. These must be deleted. The special variable names `FREQ` and `FREQUENCY` should be used in the `DATA` statement to denote frequencies of observations.
- The method of transforming variables within the `DATA` statement has changed in version 2. All transformations must be re-coded following the new syntax (described earlier in this chapter and in a later chapter). Additionally, the method of dropping or keeping variables within the `DATA` statement has changed. An example of the old syntax is

```
DATA
  v1  FIELD 1  DROP < 0
  v2  FIELD 2  ADD 10 MULTIPLY 2
  v3  FIELD 3  KEEP >= 24
  v4  FIELD 4  SUBTRACT 10 POWER 3 DROP <= 1
END     {data}
```

and the corresponding new syntax is

```
DATA
  v1  FIELD 1  DROPIF v1 < 0
  v2  FIELD 2  = (v2 + 10)*2
  v3  FIELD 3  KEEPIF >= 24
  v4  FIELD 4  = (v4 - 10)^3  DROPIF v4 <= 1
END     {data}
```

RUNNING AN *MLE* PROGRAM

Introduction

mle programs are usually run by typing `mle` followed by the name of the program file on the DOS or Unix command line. The ***mle*** interpreter will then read in the program file and execute each statement as it is encountered. If ***mle*** encounters any errors in the program, an error message is printed and execution terminates. Warning messages are printed from ***mle*** without termination.

If `mle` is typed at the command line, but without the name of the program file, you will be asked for the name of the program file. Here are some typical examples of how the `mle` is used:

```
c:\test> mle analysis.mle                Runs mle on the file analysis.mle.
c:\test> mle -v test.mle                 Runs mle on the file test.mle.  The verbose option is set.
c:\test> mle -p test.mle                 Parses the file test.mle and reports any syntax errors.
C:\test> mle                             mle will request the input file name.
MLE Program file to run? test.mle
```

If you type an erroneous command line option, or the file is not recognized by *mle* the following command line synopsis is printed:

```
c:\test> mle -z analysis.mle             There is no -z option.
Error: File "-z" does not exist

Usage: mle [-v] [-p] [-i] [-dd] [-ds] [-dp] [-di] [-dl] [-d #] [mlefile]
  -v sets verbose on. Iteration histories are printed
  -p only parses the mle file
  -i runs mle interactively
  -dd turns on data debugging
  -ds turns on symbol table debugging
  -dp turns on parser debugging
  -di turns on integration debugging
  -dl turns on likelihood debugging
  -d sets debugging to level #
  mlefile is the name of the file with the program

Usage: mle -h [name1 name2 . . . .]
  help for PDFs, functions, symbols, parameter transforms
  -h matches words exactly, -H searches within words

Usage: mle -pn n1 n2 . . . .
  parses n's and returns values and type
```

Command line options

The behavior of ***mle*** can be changed by using command line options. A list of valid command line options is given in Table 1. A particularly useful command line parameter is `-p` (parse only) which tell ***mle*** to parse the program and report any errors in the grammar. The statements within the program are not executed. Another very useful option is the `-v` (verbose)

Table 1. Command line options.

Option	Description
`-v`	Sets `VERBOSE` to `TRUE` so that an iteration history and other information is printed to standard output.
`-h` `-h <name>`	Help. Provides rudimentary information about PDFs, functions, variables, constants, reserved words, and parameter transforms. When <*name*> is replaced by a PDF name, a transformation name, a function, or a predefined variable, a brief help message is given. If <*name*> is not a known topic, a list of topics is printed.
`-H <name>`	Help. Provides rudimentary information like `-h`, but matches anything that contains the string `<name>`. If <*name*> is not given, a very long list of all help messages will be given.
`-i`	Runs ***mle*** interactively. That is, commands are typed directly in from the keyboard. Using interactive mode is particularly useful for using ***mle*** as a probability calculator (see text). Currently, this option only works in DOS.
`-p`	Sets the internal variable `PARSE` to `TRUE`. The program file is parsed for errors.
`-pn # ...`	***mle*** supports reading numbers in unusual formats (dates, times, Roman, etc.). This command line option provides a way to test the way number strings are parsed and converted into real numbers or integers.
`-dd`	Sets the internal variable `DEBUG_DATA` to `TRUE`, which turns on data debugging. When set, details are printed as each observation is read into a data set.
`-ds`	Sets the internal variable `DEBUG_SYM` to `TRUE`, which turns on symbol table debugging. Information is printed to standard output whenever variables and symbols (including internal variables) are created or destroyed.
`-dp`	Sets the internal variable `DEBUG_PARSE` to `TRUE`, which turns on parser debugging.
`-di`	Sets the internal variable `DEBUG_INT` to `TRUE`, which turns on debugging for the integration routines.
`-dl`	Sets the internal variable `DEBUG_LIK` to `TRUE` so that parameter estimates and a likelihood is written to standard output for every likelihood evaluation.
`-d #`	Sets the internal variable `DEBUG` to the value set by #. When # is greater than zero, additional information is printed out. At values over 10, an enormous amount of output is generated. A useful value is 5. A value of 0 turns off debugging.

option, which tells ***mle*** to provide periodic status reports while it is running the program and estimating parameters. Among other things, the status report prints out the likelihood and parameter values at each iteration.

Help options

mle predefines a large number of functions, variables, constants, and reserved words. The `-h` (help) option provides short summaries of ***mle*** language parts, PDFs, and concepts. Typing `mle -h` yields

```
Type mle -h <keyword> to match keywords exactly.
Type mle -H <keyword> to match partial keywords.

 mle -h MLE gives a program outline.
 mle -h PROCEDURES lists procedures.
 mle -h PDFS lists PDF types.
 mle -h FORMS lists parameter forms.
 mle -h HAZARD gives an example of a hazard specification.
 mle -h SYMBOLS lists pre-defined variables.
 mle -h NUMBERS lists number formats.
 mle -h FUNCTIONS lists simple functions,
Help is available for the following types of functions/expressions:
IDENTIFIER     ARRAY          DATA           DERIVATIVE     FINDMIN
FINDZERO       FUNCTION       IF             INTEGRATE      LEVEL
LEVELDELTA     PARAM          PDF            POSTASSIGN     PREASSIGN
PRODUCT        QUANTILE       SUMMATION

Help is available for the following statements:
ASSIGNMENT, BEGIN, DATA, FOR, IF, MODEL, PROCEDURE, REPEAT, WHILE
```

It is particularly useful for printing out a list of intrinsic parameters for PDFs. For example, typing `mle -h weibull` yields:

```
WEIBULL Distribution
4 Time variables: t(open), t(close), t(left trunc), t(right trunc)
Exact failure when t(open)=t(close)
t(open) and t(close) can SHIFT
Range:   t: (Time)  0 <= t < +oo
2 intrinsic parameters:
  a: (Scale)  0 < a < +oo
  b: (Shape)  0 < b < +oo
a is the characteristic life ~= 63.2th % in units of a
f(t) = S(t)h(t);  S(t) = exp[-(t/a)^b];  h(t) = [b*t^(b-1)]/(a^b)
mean = a*Gamma[(b+r)/b]; var = (a^2)*Gamma[(b+2)/b]-{Gamma[(b+r)/b]}^2
mode = a(1-1/b)^(1/b) for b>1; mode = 0 for b<=1; median = a*log(2)^0.5
   Gamma(x) is the gamma function
Covariate effects may be modeled on the hazard
```

which shows that there are two intrinsic parameters. Note that equations are given for the probability density, survival function, or hazard function. At least one of these is given for other PDFs as well.

Here is another example: `mle -h pi`

```
Symbol: PI{REAL CONST} = 3.14159265359
```

And, a thrid example: `mle -h besseli`

```
Function BESSELI(x1, x2)
 returns the modified Bessel fcn I (integer order x1) of real x2
```

You get lists of related keywords in mle. For example, `mle -h FUNCTIONS`, will list all of the intrinsic simple functions, and `mle -h SYMBOLS` which lists all variables in the symbol table. Typing `mle -h function | more` is a useful way to examine all ***mle*** intrinsic functions because the `more` program will stop after each page of output.

The `-H <name>` option is similar to the `-h` option except that any function, variable, constant, or reserve word that includes `<name>` as some part of the reserve word is printed. The `-H` option is particular useful when you cannot the exact name for some keyword. Thus, `mle -H integra` lists all keywords with the string "integra":

```
INTEGRATE v (expr1, expr2) expr3 END
INTEGRATE v (expr1, expr2, expr4) expr3 END
 v is the variable of integration.
 expr1 is evaluated for the lower limit of integration.
 expr2 is evaluated for the upper limit of integration.
 expr3 is the integrand, and may reference v.
 expr4 is an optional convergence criterion

 INTEGRATE_METHOD = I_TRAP_CLOSED uses closed trapazoidal integration
 INTEGRATE_METHOD = I_TRAP_OPEN uses open trapazoidal integration
 INTEGRATE_METHOD = I_SIMPSON uses open simpson integration
 INTEGRATE_METHOD = I_AQUAD (default) uses adaptive quadrature integration
 INTEGRATE_N is the number of iterations (default: 100)
 INTEGRATE_TOL is the convergence criterion (default:  1.0E-0006)

INTEGRATE_METHOD{INTEGER} = 3
INTEGRATE_N{INTEGER} = 100
INTEGRATE_TOL{REAL} = 0.00000100000
```

Other options

A number of command line options assist in debugging models, data files, program options, numerical methods, and the *mle* program interpreter itself. The `-dl` option is useful for examining likelihoods every time a complete likelihood is computed. More advanced debugging options assume some familiarity with the internal workings of parsers, symbol tables, and an advanced understanding of likelihood estimation. The `-d #` option, in particular, generates a variety of debugging messages. Details down to individual likelihoods (i.e. each observation) are generated with `-d 10`. At `-d 11`, the likelihoods produced by each subexpression of a model for each observation is printed. The `-di` option offers help with debugging problems of numerical integration in *mle*.

mle supports many formats for numbers. Each number begins with a numeral, but can contain additional symbols to specify different meanings. A full discussion of the number formats is given in the data chapter. You can test the way in which *mle* reads numbers by using the `-pn` option. The command line `mle -pn 8x3017 22'16" 12k` returns

```
"8x3017" is the integer 1551
"22'16"" is the real 0.0064771107796
"12k" is the real 12000.000000000
```

The debugging and help options send output to the screen (or standard output device). The standard DOS and Unix redirection symbols ">" and "|" can be used to redirect the output to other devices. For example, the command `mle -d 25 test.mle > test.dbg` will create a (possibly large) file called test.dbg. The output file specified within the test.mle program will not be affected.

On DOS computers *mle* can be run interactively using the `-i` command line option. When run interactively, commands are typed directly into the command line. This option is particularly useful when *mle* is used as a "calculator", which is described in the next chapter.

DATA SETS

Introduction

As a first step in parameter estimation, a *data set* must be read in or created. This chapter discusses aspects of creating a data set, including

- How to read a data set into ***mle***
- The way data files should be set up
- How to transform variables
- How to drop observations
- The number formats recognized by ***mle***
- An example of creating a data file

Data sets are read into ***mle*** from an input file. They consists of at least one, and usually many, *observations*. Each observation is a collection of one or more *variables*. The ***mle*** data statement defines how observations are to be read from the input file. The data statement also has mechanisms for doing transformations as the data are being read. In the current implementation of ***mle*** the transformations and other data manipulations provided by the data statement are not particularly powerful, but they are suitable for most applications. Other programs (spreadsheets or database managers, for example) can be used for complicated data transformations, and the resulting data set can be then used by ***mle***.

Reading data from a file

Naming the data file

Data sets are created using the `DATA` statement. The data statement typically works by reading observations from the data file. This file must be named and opened using the procedure `DATAFILE()`. The value passed to `DATAFILE()` is usually defined near the top of the program, before the `DATA` statement, as in the example in Chapter 1. The data statement begins with the word `DATA` and is terminated by an `END`. So, if the name of the data file is `MYDATA.DAT`, you must include the

statement `DATAFILE("MYDATA.DAT")` prior to the `DATA` statement. Full path names are permissible: you might call the `DATAFILE` procedure as `DATAFILE("C:\STATS\MLE\BONES\DATAFILE.DAT")`.

The DATA statement

The `DATA...END` statement reads in the data file. Within the `DATA...END` is a sequence of one or more variable names. The grammar used for specifying each variable is:

<*variable name*> [`FIELD` x [`LINE` y]] [= <*expr*>] [`DROPIF` <*expr*> | `KEEPIF` <*expr*> ...]

Variable name: Variables names begin with a letter and can then contain any combination of letters, numbers, the underscore, and period characters. A variable name may be up to 255 characters long and all characters are significant. Examples of valid variable names are: `LAST_ALIVE`, `VARIABLE_14` , `A_REALLY_LONG_VARIABLE_NAME`, and `A`. Variable names are not case sensitive so that `abc` is the same as `ABC` and `aBc`.

In the current version of ***mle***, all variables created in the `DATA...END` statement are defined to be type *real*. This is so even if the number format suggests an integer. Integer values will be read in and converted to real number values. Text strings can exist within a text file, but must not be assigned to a variable.

mle pre-defines many built in constants and variables, so you should avoid variable names that exist for some other purpose such as an ***mle*** constant (a list of all variables appears in a later chapter). Likewise, ***mle*** uses the period as an internal delimiter for some purposes. Conflicts might arise if your variable names contain a period; you are free to use periods, but an underscore might be a better choice.

Field: The term *field* refers to which column within an input file a variable is found in. In the `hammes.dat` file used in Chapter 1, four fields (or columns) existed in the input file. The field specifier must be a positive integer constant.

Line: Sometimes observations are located across multiple lines. An example might be times to first birth for a married couple in which female characteristics appear on the first line and the male characteristics occur on the second line. When the `LINE` keyword is used, e.g. `LINE 2`, ***mle*** keeps track of the maximum number of lines specified this way. Then, *all* observations are assumed to have the maximum number of lines. If the observations each take but one line, the statement `LINE 1` may be dropped—one line per observation is assumed as the default situation. The line specifier must be a positive integer constant.

The remaining specification provides ways of transforming variables and dropping (or keeping) observations. The next several sections discuss transformations and gives additional examples of declaring variables in the `DATA` section.

Dropping or keeping observations

A series of statements to drop (or keep) individual observations from the input file can be specified as the last items in a variable declaration within the `DATA` statement. Here are some example of this:

```
DATAFILE("test.dat")
my_drop_value = 100
DATA
  first_time      FIELD 3  DROPIF first_time <= 0
  missing_data    FIELD 4  DROPIF missing_data <> 1
  last_time       FIELD 1  KEEPIF last_time > 0
                           DROPIF (last_time == INFINITY) OR (first_time < last_time)
  alt_missing     FIELD 5  KEEPIF alt_missing == missing_data
END
```

The DROPIF keyword specifies that a condition will be tested; if the condition is true, then the entire observation will be dropped. The first DROPIF statement here specifies that the entire observation is to be dropped if first_time is less then or equal to zero. The KEEPIF keyword is like DROPIF except that the observation will be kept if the condition is true, and dropped otherwise. The formal grammar is KEEPIF <*bexpr*> and DROPIF <*bexpr*>, where <*bexpr*> is a boolean expression. A boolean expression is one that evaluates to true or false. Typically, boolean expressions use relational operators (>, >=, <, <=, ==, <>) and boolean operators (NOT, AND, OR, XOR). Functions that return boolean values can be used as well.

Multiple KEEPIF and DROPIF statements can be used for a single variable. As ***mle*** reads in variables, each condition is tested in sequence, until the end of the tests are reached or the observation deemed dropped (that is, boolean short-circuiting will be used to drop variables at the first opportunity). The third example is a test that keeps the observation if last_time is greater then zero; the second test will examine if the value is equal to INFINITY (a built-in constant) or less than first_time, and drop the observation if either condition is true. Then, if the variable is to be dropped, the *entire* observation is dropped. Note that the value of other variables in the current observation may be used in a DROPIF and KEEPIF statement.

Frequency of observations

Data variables with either the name FREQUENCY or FREQ are taken as a field of frequencies for each observation. (If both variable names are used, FREQUENCY is taken as the frequency variable). For example:

```
DATAFILE("test.dat")
DATA
  frequency       FIELD 1 DROPIF frequency <= 0
  start_time      FIELD 2
  last_time       FIELD 3
END
```

will take the first field in "test.dat" as the frequency for each observation. The maximizer will automatically use the frequency variable as a count of repeated observations.

Transformations

A number of simple data transformations may be made within ***mle***. The transformations are done while the data are being read from the input file. Examples of transformations are:

```
DATA
  event_time   FIELD 5 = (event_time - 1900)*365.25  DROPIF event_time < 0
  direction    FIELD 6 = COS(direction)
  winglength   FIELD 8 = LN(winglength/2.25)
  estage               = 3.7 + winglength*12.76 + winglength^2 * 1.14
END
```

Transformations begin with '=' which is then followed by an expression. Expressions are discussed in more detail in a later chapter. Basically, expressions in *mle* are similar or identical to expressions found in other computer languages and spreadsheets.

In the first variable declaration of the example, `event_time` is read in from the input file. That initial value of `event_time` is then used in the transformation, and a new value of event_time is computed as `(event_time - 1900)*365.25`. This result is assigned back to `event_time`. Following that, the `DROPIF` statement will conditionally decide whether or not the observation is to be dropped.

Variables are read in the same order in which they are defined. This is true even if they are read over several lines. Once a variable is defined, its value can be used in later transformations. Then, when reading in the data file, *mle* will take the value of that variable for the current observation for use in the later transformation. An example might be:

```
DATA
  subject_id   FIELD 1  DROPIF subject_id =1022 DROPIF subject_id = 3308
  births       FIELD 6  DROPIF births = -1
  miscarriages FIELD 8  DROPIF miscarriages = -1
  abortions    FIELD 9  DROPIF abortions = -1
  pregnancies  = births + miscarriages + abortions  KEEPIF pregnancies > 0
END
```

This data statement will read `subject_id`, then `births`, then `miscarriages` and then `abortions`. These variables will then be added together and assigned to the variable `pregnancies`. An observation will be dropped if any of `births`, `miscarriages`, or `abortions` are negative one (in this case, the "missing" code), or if two particular `subject_ids` are found, or if `pregnancies = 0`.

Creating dummy variables

Dummy variables can be easily created. Suppose you are measuring the length of some study animal. You want to create four dummy variables for the length range short [0 to 30 mm), medium [30 to 40 mm) long [40 to 50 mm) and very long [50+ mm):

```
DATA
  length       FIELD 5 DROPIF length <= 0
  is_short     = IF length < 30 THEN 1 ELSE 0
  is_medium    = IF (length >= 30) AND (length < 40) THEN 1 ELSE 0
  is_long      = IF (length >= 40) AND (length < 50) THEN 1 ELSE 0
  is_verylong  = IF length >= 50 THEN 1 ELSE 0
END
```

Skipping initial lines in the data file

Data files may have initial descriptive lines at the top that must be skipped. The `INPUT_SKIP` controls how many lines to skip in a data file. For example, if the first four lines must be skipped, the line

```
INPUT_SKIP = 4
```

should appear before the `DATA` statement. It will direct *mle* to discard the first four lines of the data file. The default value is zero so that no lines are skipped.

Delimiters in the data file

Data files consist of a series of text elements separated by one or more *delimiters*. One or more delimiters must appear between each record within a data file. The delimiters define the fields within each line in which variables reside. By default, the characters space, tab, and comma are treated as delimiters. You can redefine the delimiters by changing the variable DELIMITERS before the DATA statement. If, for example, you wanted the colon and semicolon character as the only valid delimiters, you would add the line:

```
DELIMITERS = ":;"
```

Creating observations without a file

Rather than reading observations from a file, observations can be created. This is useful for simple simulation programs using the random number generators in ***mle***. To create variables, simply set the variable CREATE_OBS to some positive number. That number of observations will be created. Here is an example:

```
CREATE_OBS = 10          {create 10 observations}
SEED(8936)               {set the random number generator seed}
DATA
  var1   = RAND  {random number from 0 to 1}
  var2   = IRAND(100, 200)
  var3   = sin(pi*RAND)
END
```

Yields the following set of data:

```
var1        var2   var3
0.46991484  157.0  0.98095861
0.76562640  117.0  0.24396827
0.80010137  173.0  0.73070002
0.92179122  139.0  0.86426399
0.43740313  197.0  0.88247371
0.01521996  136.0  0.09665617
0.46592947  136.0  0.39891672
0.02549209  198.0  0.78123339
0.49985020  185.0  0.36516675
0.83997806  193.0  0.48128269
```

Printing observations and statistics

Some other variables can be used to fine-tune the DATA statement.

The variable PRINT_DATA_STATS, when set to TRUE, prints summary statistics for each variable, including the mean, variance, standard deviation, minimum and maximum. The default is TRUE, so this report can be suppressed with PRINT_DATA_STATS = FALSE.

When PRINT_OBS is set to TRUE, each observation is printed to the output file. The report is printed *after* all transformations have been done. The default value is FALSE, so you must have the statement PRINT_OBS = TRUE to print the observations.

The variable `PRINT_COUNTS`, when set to `TRUE`, prints out how many lines were read from the input file, how many observations were kept, and how many observations were dropped. The default value is `TRUE`, so these reports can be suppressed with `PRINT_COUNTS = FALSE`.

Number formats

The ***mle*** language is primarily designed for operations on numbers. With this in mind, a wide variety of number formats, including some with automatic conversions, are supported. The standard formats for real and integer numbers are recognized, so that "3.14159", "-12.14" and "0.001" are read as would be expected. Real numbers must have a leading zero, so ".23" is not valid but "0.23" is. Real numbers can be in scientific notation so that "2.1E-23", "0.3E12", "-1e4", "12345e-67 are valid numbers.

Table 2. Standard number formats.

Format	Examples	Conversion	Result
d	1, 200		integer
d.d, d.	3.1415, 3.		real
ds, *-ds*, *d.ds*, *-d.ds*,	14%, 23.7M, 45.7da, 2n, 2.418E	Metric suffix (see Table 3)	real
*d*E*d*, *d*E-*d*, *d.d*E*d*, *d.d*E-*d*, *d*.E*d*, *d*.E-*d*	3e23, 511E-10, 31.416e-1, 7.0E-10, 12.e-6, 1.45E-3, 1.0E0	Standard exponential format. $x\text{E}y \Rightarrow x \times 10^y$	real
0R*v*	0RXLVII, 0rMXVI, 0rmdclxvi	Roman numerals to integer	integer
*d*X*y*	2x1001 (binary), 8X3270 (octal), 16xA4CC (hex), 32x3vq4h (base 32).	Converts *y* from base *d* (from 2 to 36) into integer.	integer
d:*d*:*d*, *d*:*d*:*d.d*, *d*:*d*, *d*:*d.d*	10:42, 14:55:32, 10:40:23.4, 16:53.2	24-hour time into hours. Hours must be 0-24.	real
d:*d*:*d*AM, *d*:*d*:*d*PM, *d*:*d*:*d.d*AM, *d*:*d*:*d.d*PM, *d*:*d*PM, *d*:*d*AM, *d*:*d.d*AM, *d*:*d.d*PM	10:42AM, 2:55:32pm, 10:40:23.4am	12-hour time with AM and PM suffixes into hours. Hours must be 0-12.	real
*d*H*d*'*d*", *d*H*d*'*d.d*", *d*H*d*', *d*H*d.d*", *d*H*d.d*"	230h16'32", 14H32'6", 100h22', 30H32.2', 0h12', 0H12'3"	Degree/hour minute, second format. Converted to real angle/time.	real
d`*d*'*d*", *d*`*d*'*d.d*", *d*`*d*', *d*`*d.d*", -*d*`*d*'*d*", *d*`, *d.d*`, *d*°*d*'*d*", *d*°*d*'*d.d*", *d*°*d*', *d*°*d.d*", *d*°, *d.d*°	230`16'32", 14`32'6", 100`22', 30`32.2', 14`, 230°16'32", 14°32'6", 270°10'0", 30°18.2', 3.4°	Degree, minute, second format, converted to radians.	real
d'*d*", *d*'*d.d*", *d*', *d.d*', *d*", *d.d*"	12'32", 166'12.9", 19', 14.7', 12", 607.3"	Minute-second and second format, converted to radians.	real
*d*_*d*/*d*	12_5/16, 3_2/3, 0_1/7	Fraction notation.	real
*d*D*d*M*d*Y	16d12m1944y, 1D6M1800Y	Date converted to Julian day	integer
*d*M*d*D*d*Y	12m16d1944y, 6M1D1800Y	Date converted to Julian day	integer
*d*Y*d*M*d*D	1944y12m16d, 1800Y6M1D	Date converted to Julian day	integer
dmmmy	14Dec1999, 30jun1961, 1MAY1944	Date converted to Julian day	integer

d is a strings of one or more positive digits; *s* is a one or two character case-sensitive metric or percent suffix (see Table 3), *v* is a string of one or more Roman numeral digits {IVXLCDM}, *y* is a string of one or more characters, *mmm* is a 3-character English month name. E.g. jan, Feb, MAR, etc. The degree character (°) is available on some hardware platforms as ASCII code 230. On many Intel platforms, holding down the <ALT> key and typing 230 on the numeric keypad gives the degree character.

The Greek letter micro (µ) is available on some hardware platforms as ASCII code 248. On many Intel platforms, holding down the <ALT> key and typing 248 on the numeric keypad gives this character.

Less common formats include numbers with metric and percent suffixes, numbers interpreted as time, numbers in an angle notation (one format that converts degrees to radians), numbers in bases from 2 to 36, Roman numerals, numbers in fraction notation, and several date formats. These formats are supported in data files as well as numeric constants within an ***mle*** program. Table 2 is a comprehensive list of formats recognized by ***mle*** , and Table 3 is a list of suffixes permissible on standard integer and real format numbers.

An example of creating a data file

The format of the data file is ordinary ASCII text, and the file can be created with any text editor. Word processors can be used to create files as well, but the results must be saved as ASCII text. Nearly all word processors provide an ASCII text option. An example of a typical data file can be seen in Chapter 1, but here we will examine a more complicated data file and write the ***mle*** program to read and process the file.

The current version of ***mle*** creates variables of type real, and attempts to read real numbers for each variable. Even so, any delimited text can appear in fields that are not assigned to variables. Consider how we would create a `DATA` statement to read the numeric values for the following file:

```
Last   First,MI     Age   Amount     More   Rate  Time
Smith  James,A      42     12000     TRUE   18%    4.2
Jones  David,J      38      8000     FALSE  12%    3.1
Connor Mary         50     11000     TRUE   19%    2.1
```

First of all, notice that there is a header on the first line of the file. This line should be discarded by setting `INPUT_SKIP=1`. From there, the data files has one line per observation, with each variable corresponding to one column. Some data files place one observation across multiple lines, so that the `LINE` option in the `DATA` statement must be used. We will not need to use the `LINE` specification here.

This file consists of seven fields delimited by space characters. Since the space character is one of the default delimiters, we do need to change the delimiters to recognize the space as such. But, since we have commas embedded in the text that should not to be taken as delimiters, we must redefine delimiters to exclude the comma and include the space (and the tab character, if necessary). The numeric values appear in fields 3, 4, 6, and 7. We do not need to do anything with fields 1, 2, and 5. Let us say that we want to convert time from years into months. Here is the complete ***mle*** code to read and process this file (but no analyses are specified):

```
MLE
  DATAFILE("THEDATA.DAT")
  PRINT_OBS = TRUE     {print out each observation}
  INPUT_SKIP = 1       {get rid of the header line}
  DELIMITERS = " "     {spaces only--treat commas as text}
  DATA
    age       FIELD 3
    amount    FIELD 4  DROP <= 0
    rate      FIELD 6    {% is a legal number suffix in mle}
    time      FIELD 7  MULTIPLY 12
  END
END
```

Table 3. Standard metric suffixes for integer and real numbers.

Suffix	Name	Conversion	Suffix	Name	Conversion
da	deka	$\times 10$	d	deci	$\times 10^{-1}$
h	hecto	$\times 10^{2}$	c, %	centi, percent	$\times 10^{-2}$
k	kilo	$\times 10^{3}$	m	milli	$\times 10^{-3}$
M	mega	$\times 10^{6}$	μ, u	micro	$\times 10^{-6}$
G	giga	$\times 10^{9}$	n	nano	$\times 10^{-9}$
T	tera	$\times 10^{12}$	p	pico	$\times 10^{-12}$
P	peta	$\times 10^{15}$	f	femto	$\times 10^{-15}$
E	exa	$\times 10^{18}$	a	atto	$\times 10^{-18}$

Running ***mle*** on this file produces the output to the screen (or standard output) since no `OUTFILE` procedure was called. Here are the results:

```
3 lines read from file THEDATA.DAT
3 Observations kept and 0 observations dropped.

NAME          age      amount        rate        time
    1  42.0000000  12000.0000  0.18000000  50.4000000
    2  38.0000000  8000.00000  0.12000000  37.2000000
    3  50.0000000  11000.0000  0.19000000  25.2000000

MEAN   43.3333333  10333.3333  0.16333333  37.6000000
VAR    37.3333333  4333333.33  0.00143333  158.880000
STDEV  6.11010093  2081.66600  0.03785939  12.6047610
MIN    38.0000000  8000.00000  0.12000000  25.2000000
MAX    50.0000000  12000.0000  0.19000000  50.4000000
```

THE MODEL STATEMENT

Introduction

The model statement is the meat of the ***mle*** programming language. It specifies the likelihood, defines parameters, and specifies which parameters are to be estimated. A complete understanding of how models are built in ***mle*** requires an understanding of the structure of the `MODEL` statement, an understanding of parameters and how they are specified, an understanding of how expressions are specified and are built into likelihoods, and an understanding of the specification for running models.

This chapter discusses the `MODEL` statement. It is assumed that you understand the basics of expressions and data types for the ***mle*** language. Chapter 1 provided much of the necessary background. This chapter covers several topics that are closely related to building typical likelihood models in ***mle***: the `PARAM` function, the `PDF` function, and the `DATA` and `LEVEL` functions.

Structure of the MODEL statement

The basic structure of the `MODEL` statement looks like this:

```
MODEL
  <expression>
RUN
  <run list>
END
```

The single *<expression>* in the `MODEL` statement defines the likelihood that is to be maximized. (Expressions are described in some detail here, other details are given in other chapters). Here is an example of a simple model for finding the two parameters for a normal PDF from interval censored observations.

```
{1}     MODEL
{2}       DATA
{3}         PDF NORMAL(topen, tclose)
{4}           PARAM  mu     LOW = 5   HIGH = 14  START = 8   END
{5}           PARAM  sigma  LOW = 0.1 HIGH = 5   START = 1.2 END
{6}          END  {pdf}
{7}       END  {data}
{8}     RUN
{9}       FULL
{10}    END
```

Everything beginning with the `DATA` function on line 2 to the `END` on line 7 is a single expression. That expression defines a likelihood. Values for the parameters `mu` and `sigma` will be found that maximize this likelihood. The likelihood in this example is the product of interval censored observations between topen and tclose, and is equivalent to

$$L = \prod_{i=1}^{N} \left[S(t_{open_i} \mid \mu, \sigma) - S(t_{close_i} \mid \mu, \sigma) \right]$$

for N observations and with $S(t)$ defined as the survival function for a normal distribution

The expression that defines the likelihood within a model statement can become much more complicated than this example. A likelihood that is made up of a more complicated expression is given in Example 1. Here the *<expression>* begins with the `DATA` function and ends with a matching `END` just before the `RUN`. Within the `DATA` function, the `MIX` function is immediately called, and the `MIX` function contains three arguments separated by commas. Each of these three arguments of the `MIX` function contains an expression. Here, we see one parameter (a mixing proportion) and two function calls: `PDF...END`. The likelihood, in symbols, is

$$L = \prod_{i=1}^{N} \left\{ p \left[S(t_{open_i} \mid \mu_1, \sigma_1) - S(t_{close_i} \mid \mu_1, \sigma_1) \right] + (1-p) \left[S(t_{open_i} \mid \mu_2, \sigma_2) - S(t_{close_i} \mid \mu_2, \sigma_2) \right] \right\}$$

for N observations and with $S_1(t)$ defined as the survival functions for a normal distribution.

Runlist

Sometimes parameters are constrained for the purpose of hypothesis testing or modifying the model. Parameters may be held constant, or fixed to the value of another parameter. These are called *fixed parameters*, and an estimate will not be found for them. The runlist in ***mle*** provides the mechanism for fixed parameters primarily to reduce models from more complicated to simpler forms. For example, in a slope function, we may have reason to believe that the slope m is one. Perhaps this is because of the nature of the physical system we are modeling. We could first fit our collection of x values to the model with parameter m free, and secondly fit it with m held constant to 1. Statistical criteria can then be used to determine whether m deviates from the value we expected it to be.

The run list defines which parameters are free and allows the user to test reduced models. The run list begins with the word `RUN` and ends with a matchin `END`. Between the `RUN` and the `END` comes a list that specifies how the model is to be run. When `FULL` is specified, all free model parameters for the model are estimated. The `REDUCE` keyword provides a mechanism to constrain parameters of the model. The `REDUCE` keyword is followed by a list of constraints. Parameters may be constrained to other parameters, to constants or to variables. More than one `REDUCE` keyword may occur in a single run list. Generically, a runlist looks like this:

```
RUN
  FULL
  REDUCE  <reduce list>
  REDUCE  <reduce list>
  ...
END
```

The *<reduce list>* is a set of one or more parameter constraints that look like assignment statements. Parameters so constrained will not be estimated. The following example includes one full and three reduced runs.

```
MODEL
  DATA
    PDF NORMAL(topen, tclose)
      PARAM mean ...
        COVAR sex  PARAM b_sex ... END
      END {param}
      PARAM stdev ... END
    END  {pdf normal}
  END  {data}
RUN
  FULL   {Runs the model with no constraints}
  REDUCE  mean = 4                {One constraint}
  REDUCE  mean = 4  b_sex = 0     {Constrains 2 parameters}
  REDUCE  mean = oldmean          {Fixes mean to another param or variable}
END
```

In this example, parameters will be estimated four times. For the first case (`FULL`) three parameters will be estimated. For the second case, the `mean` parameter will be constrained to 4 so that only two parameters will be estimated. For the third case, only on parameter is free to be found, and for the fourth case, two parameters are free.

mle provides a mechanism for accessing results from previous runs within the same model. Thus, in the previous example, the parameter `mean` and `stdev` are really called `mean.1` and `stdev.1` when the full model is run. Likewise, the parameters are called `mean.4` and `stdev.4` for the last run. For a reduce statement like `REDUCE mean = stdev`, ***mle*** will assume the parameters refer to the current run. That is, ***mle*** treats them as `REDUCE mean.1 = stdev.1` (assuming this is the first entry of the run list).

Expressions used in MODEL statements

Expressions are used in many ways within ***mle***, so that you should become thoroughly acquainted with expressions before attempting to develop ***mle*** programs. For example, the likelihood within a `MODEL` statement is a single (sometimes complicated) expression. Expressions are used to define limits of integration, summations, and products, they can be used to define `start`, `high`, `low`, and `test` values for parameters, and many other things. The right-hand side of an assignment is an expression, as are data transformations in the `DATA` statement. Boolean expressions are used in `IF` statements, `DROPIF` and `KEEPIF` and elsewhere.

A brief summary of the types of functions defined in ***mle*** is given in Table 5. At the simplest level, an expression in ***mle*** can be a numerical constant or a variable name. More complex expressions consist of algebraic operators (*,^,+, etc) and function calls each with zero or more arguments. Most functions in ***mle*** are simple functions with a fixed number of arguments, for example: `PERMUTATIONS(x, y)`, `ARCSIN(x)`, `ABS(x)`, `MIX(p, x, y)`.

A second class of functions are more complex, and have a more complicated syntax. These functions begin with a keyword, and end with an `END`. Examples of some of these functions are the `PARAM...END` function, `DATA...END` function (not to be confused with the `DATA END` *statement* described in a previous chapter), the `PDF...END` function, the `INTEGRATE a (b, c)...END` function, and the `IF THEN...ELSE...END` function.

Suppose you want to integrate $\sin(x^2 + 2x)$ from $-\sqrt{\pi}$ to $\sqrt{\pi}$. Here is an example of how that could be coded: `INTEGRATE x (-SQRT(PI), SQRT(PI)) SIN(x^2 + 2*x) END`. (The function evaluates to ≈ -1.525). Here it is with comments:

The MODEL statement

```
MODEL     {mixture of two normal distributions}
  DATA
    MIX(
      PARAM   p  LOW = 0  HIGH = 1  START = 0.5 END
    ,
      PDF NORMAL(topen, tclose)
        PARAM  mu1     LOW = 5   HIGH = 14  START = 8 END,
        PARAM  sigma1  LOW = 0.1 HIGH = 5   START = 1.2 END
      END  {PDF}
    ,
      PDF NORMAL(topen, tclose)
        PARAM  mu2     LOW = 0   HIGH = 6  START = 2 END,
        PARAM  sigma2  LOW = 0.01 HIGH = 5   START = 1.2 END
      END  {PDF}
    )  {mix}
  END {data}
RUN
  FULL
END       {model}
```

Example 1. Model specification for estimating a mixture of two normal distributions with interval censored observations.

```
INTEGRATE x  (      {x is the variable of integration}
  -SQRT(PI),        {This is the lower limit of integration}
   SQRT(PI)         {This is the upper limit of integration}
             )      {Close of the argument list}
    SIN(x^2 + 2*x)  {The function to be integrated}
END                 {End of the integrate function}
```

Any of the predefined probability density functions can be used as part of an expression. For example, if you wanted the area between 8 and 12 under a normal distribution with μ=10 and σ=3, you could write that expression as

```
PDF NORMAL(8, 12) 10, 3 END
```

The PARAM *function*

mle has a general method for defining all parameters to be used in a likelihood model.[5] The PARAM function defines a parameter and its characteristics. The function should only be used within a MODEL statement. When models are "solved", *free parameters* are estimated by iteratively plugging new values in for those parameters until the values that maximize the likelihood are found. In other words, free parameters are values that are to be estimated by ***mle*** —they are the unknowns in likelihood models. If the parameter is not constrained to some fixed value in the RUN part of the model statement, ***mle*** will estimate the value of that parameter.

In the simplest case, parameters are specified as

```
PARAM  <p> HIGH = <expr>.LOW = <expr> START = <expr> TEST = <expr>..FORM = <formspec> END
```

where <*p*> is the name chosen for the parameter. The keywords HIGH, LOW, START, and TEST specify characteristics for the parameter. HIGH and LOW specifies the minimum and maximum value allowed for the parameter. ***mle*** will not exceed these values while trying to maximize the likelihood. START tells the maximizer what vaule to start with. TEST denotes the value against which to test the parameter for significance. By default, TEST is zero. The TEST value does not change anything about how the parameter is maximized. It is only used for a *t*-test as the parameter is being written to the output file.

[5] The word *parameter* is used in a very specific way, as defined in Chapter 1. Parameters are the quantities to be estimated in a likelihood model

The PARAM function allows covariate effects (and their associated parameters) to be modeled within the parameter statement. This is done as follows:

```
PARAM  x HIGH = <expr> LOW = <expr> START = <expr> TEST = <expr>  FORM = <formspec>
  COVAR <expr>  PARAM  z .HIGH = ... END
  ...
END  {param}
```

With covariates, the <*expr*> following COVAR is a covariate effect. Typically this is a variable like age, sex, income, etc. The effect of the covariate is multiplied by the value of the PARAM function. The way in which covariates and parameters are modeled is discussed in more detail below.

Here is an example of a likelihood hand-coded for an exponential PDF for exact failure times. PARAMs and built-in simple functions, and algebraic expressions are all shown in this likelihood:

```
MODEL
  DATA
    PARAM lambda LOW = 0 HIGH = 1 START = 0.23 END * EXP(-lambda * t)
  END
RUN
  FULL
END
```

Notice that lambda is first defined as a parameter, and thereafter is used as an ordinary variable. As ***mle*** iteratively seeks a solution, new values of lambda will be tried. As the likelihood itself is being computed, the PARAM function will simply return the current estimate of lambda.

An alternative way to code this example is to define the parameter first and assign it to another variable:

```
MODEL
  PREASSIGN
    lam = PARAM lambda LOW = 0 HIGH = 1 START = 0.23 END
    DATA
      lam*EXP(-lam*t)
    END  {data}
  END  {preassign}
RUN
  FULL
END {model}
```

The PREASSIGN function is described in another chapter.

In Example 1, five parameters are defined, two each for the two PDF functions and one parameter that was added for the first argument to the MIX function call.

Typically, parameters are defined for the intrinsic parameters of a PDF function. For example, the normal PDF has two intrinsic parameters μ and σ. The first parameter specified in the parameter list will be treated as μ. The second will be treated as σ. How can you know the proper order for parameters? Generally location parameters appear first (and are usually denoted *a* in this manual), scale parameters are second and shape parameters are third. Even so, you can get a quick synopsis of each type of PDF by using the -h option from the command line, e.g.: `mle -h SHIFTWEIBULL`

Parameters are also used to model effects of covariates on other parameters. Here is an example in which two parameters, used in place of some fixed values of μ and σ for a normal distribution, are defined with two covariate parameters, each:

Table 4. Forms and transformations for parameters.

Form	Parameter (p'), covariates ($\mathbf{x}_i$), covariate parameters ($\boldsymbol{\beta}$), and the value returned by the PARAM function (p_i)	Notes
NUMBER	$p_i = p$'	Default when no COVARs are modeled.
ADD	$p_i = p' + \mathbf{x}_i\boldsymbol{\beta}$	Must be used with care when the resultant parameter is constrained to positive values because p_i might take on negative values for some combinations of $\mathbf{x}_i\boldsymbol{\beta}$
INVERT	$p_i = 1/(p' + \mathbf{x}_i\boldsymbol{\beta})$	The denominator must not be zero.
INVADD	$p_i = 1/p' + \mathbf{x}_i\boldsymbol{\beta}$	p' must not be zero.
INVMULTIPLY	$p_i = \mathbf{x}_i\boldsymbol{\beta}/p'$	p' must not be zero.
INVLOGLIN	$p_i = \exp(\mathbf{x}_i\boldsymbol{\beta})/p'$	p' must not be zero.
DIVIDE	$p_i = p'/\mathbf{x}_i\boldsymbol{\beta}$	$\mathbf{x}_i\boldsymbol{\beta}$ must not be zero.
POWER	$p_i = p'^{\mathbf{x}_i\beta}$	
POWEREXP	$p_i = p'^{\exp(\mathbf{x}_i\beta)}$	
EXPADD	$p_i = \exp(p' + \mathbf{x}_i\boldsymbol{\beta}) = \exp(p')\exp(\mathbf{x}_i\boldsymbol{\beta})$	Constrains p_i to positive values for all p' and $\mathbf{x}_i\boldsymbol{\beta}$.
MULTIPLY	$p_i = p' \times \mathbf{x}_i\boldsymbol{\beta}$	A multiplicative specification.
EXCESS	$p_i = p'\exp(1 + \mathbf{x}_i\boldsymbol{\beta})$	
LOGLIN	$p_i = p'\exp(\mathbf{x}_i\boldsymbol{\beta})$	This is a common specification, especially for parameters that are interpreted as hazards. When p' is constrained positive, the p_i will also be positive. Like EXPADD but $p'_{expadd} = \exp(p'_{loglin})$. LOGLIN is the default specification whenever a COVAR is defined.
LOGISTIC	If ALTERNATE_LOGISTIC = FALSE, $p_i = 1/[1 + \exp(p' + \mathbf{x}_i\boldsymbol{\beta})]$. If ALTERNATE_LOGISTIC = TRUE, $p_i = \exp(p' + \mathbf{x}_i\boldsymbol{\beta})/[1 + \exp(p' + \mathbf{x}_i\boldsymbol{\beta})]$	Frequently used for parameters that are interpreted as probabilities because, for all values of $p' + \mathbf{x}_i\boldsymbol{\beta}$, p_i will be constrained from zero to one. The alternative forms are related to each other as $p'_{form1} = 1 - p'_{form2}$
LOGIT	$p_i = \ln[\exp(p' + \mathbf{x}_i\boldsymbol{\beta})/(1 + \exp(p' + \mathbf{x}_i\boldsymbol{\beta})]$	This specification is useful when p_i can take on any value from $-\infty$ to ∞ and $p' + \mathbf{x}_i\boldsymbol{\beta}$ is a probability.

```
PDF NORMAL(topen tclose)
  PARAM mean  LOW = 100  HIGH = 400  START = 270  TEST = 0  FORM = LOGLIN
    COVAR  sex    PARAM b_sex_mu     LOW = -2  HIGH = 2  START = 0 END
    COVAR  weight PARAM b_weight_mu  LOW = -2  HIGH = 2  START = 0 END
  END
  PARAM stdev     LOW = 0.1  HIGH = 100 START = 20  FORM = LOGLIN
    COVAR  sex    PARAM b_sex_sig     LOW = -2  HIGH = 2  START = 0 END
    COVAR  weight PARAM b_weight_sig  LOW = -2  HIGH = 2  START = 0 END
  END
END
```

In this example, the first parameter of the normal distribution (μ) has two covariates and their corresponding parameters modeled on it. The exact specification of how covariates and their parameters are modeled depend on the FORM of the intrinsic parameter. In the example, the FORM = LOGLIN specifies that a log-linear specification is to be used. The log-linear specification is $\mu_i = \mu'\exp(\mathbf{x_i}\boldsymbol{\beta})$, where μ' is the estimated intrinsic parameter (mean in this case). Thus, for the ith observation, the μ parameter of the normal distribution will be constructed as: μ_i=mean×exp(sex$_i$×b_sex + weight$_i$×b_weight). The second parameter, stdev, has the same two covariates modeled on it, but the parameter names are (and must be) different from the parameters modeled on mean.

For some forms, the parameter itself is transformed. For example, when a parameter is a probability (as it is for the MIX function in Example 1) the parameter can be defined as:

```
PARAM p  LOW = -999  HIGH = 999  START = 0  FORM = LOGISTIC END
```

The logistic transformation permits the parameter p to take on any value from negative infinity to infinity, but the resulting value passed used by the likelihood will be constrained to the range (0, 1). In other words, ***mle*** will estimate a parameter over the range –999 to 999, but before that parameter is used in computation, it will undergo a logistic transformation as $p = 1/[1 + \exp(p')]$, so that the value of p will be a probability. ***mle*** currently provides a limited number of specifications for how parameters and covariates are modeled (Table 4). Even so, this mechanism for modeling covariates on any parameter is extremely general and provides the basis for building unique and highly mechanistic (Box et al. 1978) or etiologic (Wood 1994) models.

Setting parameter information

Four characteristics may be set for each parameter in addition to the FORM. They are: 1) the highest possible value that can be tried for the parameter, 2) the lowest possible value that can be tried for the parameter, 3) the starting guess to help ***mle*** out from the start, and 4) a test value against which the parameter will be tested when standard errors are computed. In the previous example, the mean parameter was constrained to the range [100, 400] and the initial guess was 270.

Use care when setting the HIGH and LOW limits. Most importantly, limits must be constrained to valid ranges for the intrinsic parameter. Thus, for the MIX mixing proportion parameter (the first of the three parameters) then, HIGH = 1 and LOW = 0, should be defined as is appropriate for a probability—unless some FORM like FORM = LOGISTIC is used to constrain the resulting parameter to between 0 and 1. Sometimes it is useful to impose narrower limits, perhaps to avoid getting hung-up at a local maximum or to solve the model more quickly. Be careful, though. Limits that are too narrow may exclude the global maximum—after all, the best parameter estimates for a set of data are presumably unknown. Excessively narrow limits may cause problems when numerical derivatives for the variance-covariance matrix are computed, as well. Also, likelihood confidence intervals will bump up and stop at the limits you set.

The TEST = xxx part of a PARAM function provides a value against which the parameter will be tested (in some reports). In a sense, the TEST value is a null hypothesis value (h_0). The test performed is $t = (\hat{p} - h_o) / SE(\hat{p})$, where $\hat{p}$ is the maximum likelihood parameter estimate and $SE(\hat{p})$ is the standard error for the parameter estimate. The *t*-test is provided for convenience only. ***mle*** does not make use of the test in any way.

The PDF functions

One of the most frequently used functions in the MODEL statement is the PDF function. The purpose of the PDF function is to specify the component of a pre-defined probability density or distribution functions. Although the name is PDF, the PDF function can return either the probability density function or specified areas under the PDF curve including the

Table 5. Brief summary of some types of functions in ***mle***.

function	Brief description
PDF	Specifies a pre-defined probability or cumulative density function. Returns the value of the density or distribution function as is appropriate to the arguments with which is was called.
INTEGRATE	Integrates a function and returns the value of the integral.
IF THEN ELSE	Evaluates a condition and returns the appropriate subfunction.
PREASSIGN	Computes a subfunction and assigns the result to a variable. Then it computes and returns the second subfunction.
POSTASSIGN	Computes the first subfunction. Then it computes the second subfunction, assigns the result to a variable, and then returns the value of the first subfunction.
PRODUCT	Iterates over some limits and takes the product of a subfunction.
SUMMATION	Iterates over some limits and sums a subfunction.
function calls	A number of ordinary mathematical functions. Example: `SQRT(x)` and `ADD(x, y)`.
quick conditionals	`ZEROIF`, `ONEIF`, `NEGONEIF`, `INFINITYIF`, `NEGINFINIF`, `NEGONEIF`: return common values conditional on simple tests.
constants, variables	Pre-defined and user defined variables can be used as functions.
PARAM	A parameter to be estimated.
DATA	The data function cycles through all observations
LEVEL, LEVELDELTA	Creates a nested or multilevel likelihood.

cumulative and survival density functions, and the hazard function. In addition, the `PDF` function can return areas or densities that are left and right truncated. The structure of the `PDF` function call is:

```
PDF <PDF name> ( <time variable1>, <time variable2>, ... )
  <intrinsic parameter 1>,
  <intrinsic parameter 2>,
  ...
  <optional HAZARD>
END
```

The *name* following `PDF` is the name of the built-in distribution. ***mle*** predefines over 60 density functions, including most well-known ones like the normal, lognormal, weibull, gamma, beta, and exponential distributions. A complete summary of built-in distribution is given in a later chapter.

Time variable list is a list of the time arguments passed to the PDF. Most univariate PDFs can take from one to four 'time' arguments.[6] In fact, these four times describe a single observation in such a way as to incorporate a number of defects in the observation process, including right censoring, left truncation, right truncation, cross-sectional observations. A description of how the four arguments combine to specify a probability are given in the section that follows. Note that the time arguments can be any expression, so that time shifts and transformations can be incorporated in this list.

[6] These are called *time* variables in the context of survival analysis; however, they may represent other measurements (length, dose, height, etc.).

Intrinsic parameter list provides specifications for the PDF's intrinsic parameters. The order that the intrinsic parameters are specified is important; it corresponds to how the PDF is defined within ***mle***. The PDFs chapter lists the order for intrinsic parameters; alternatively, the command line `mle -h` can be used to determine the proper argument order. Note that any expression can be used for an intrinsic parameter. That is, you do not need to use a `PARAM` function for the intrinsic parameters, although this is the most common use. Here is an example in which the location parameter is fixed to a constant for a shifted lognormal distribution:

```
PDF SHIFTLOGNORMAL ( tooth_eruption_age )
  9,  {shift the time back to conception}
  PARAM location LOW = 1 HIGH = 4 START = 2.5 END,
  PARAM scale LOW = 0.0001 HIGH = 3 START = 0.9 END
END
```

PDF time arguments

Most PDFs can have as few as one and as many as four time arguments specified. They are: t_u, the last observation time before an event; t_e, the first observed time after the event; t_α, the left truncation time for the observation or the PDF; and t_ω, the right truncation time for the observation or the PDF. Understanding how these four times act on the `PDF` statement is critical to creating the desired and proper likelihood.

PDFs contribute to likelihoods in a number of ways. In survival analysis, for example, the likelihood for an exact failure is given by the value of the PDF at the exact point of failure. For a right censored observation, the likelihood is given by summing up (integrating) all possible PDF values from the last observation time until the maximum possible time. The likelihood for a cross-sectional "responder" is the integral from zero to the time of first observation. Table 6 lists the likelihoods that result from the four time variables for different conditions. For example, when t_u=t_e or when only one time variable is specified, ***mle*** returns the density at t_u. This is the desired likelihood for an exact failure. Likelihoods for right and interval censored observations, with and without left and right truncation are given in Table 6.

The Hazard parameter

For most parametric distributions (like the normal or lognormal distributions) the hazard function does not take on a simple or closed form. For this reason, most studies have modeled the covariates as acting on the failure time for these distributions. Nevertheless, there is no inherent reason why hazards models cannot be constructed using distributions without a closed form for the hazards functions. Most of the PDFs included in ***mle*** provide a general mechanism for covariates to be modeled as affecting the hazard of failure, rather than (or in addition to) affecting intrinsic parameters. Here is an example:

```
PDF NORMAL(topen tclose)
  PARAM mean   LOW = 100  HIGH = 400  START = 270  TEST = 0  FORM = LOGLIN END,
  PARAM stdev  LOW = 0.1  HIGH = 100 START = 20 END,
  HAZARD COVAR  sex     PARAM b_sex     LOW = -2  HIGH = 2  START = 0 END
         COVAR  weight  PARAM b_weight  LOW = -2  HIGH = 2  START = 0 END
END
```

The covariates `sex` and `weight` are modeled to effect on the hazard of failure. Parameters `b_sex` and `b_weight` provide estimates of the effect.

Table 6. Likelihoods returned by PDF for one, two, three, and four time variables under different conditions.

Example	When	Class	Resulting Likelihood
LNNORMAL(t_e)		Exact failure at t_e	$L = f(t_e)$
LNNORMAL(t_u, t_e)	t_u=t_e	Exact failure at t_u=t_e	$L = f(t_u) = f(t_e)$
LNNORMAL(t_u, t_e)	t_e=○○ $t_e < t_u$	Right censored or cross-sectional non-responder at t_u	$L = \int_{t_u}^{\infty} f(z)dz = S(t_u)$
LNNORMAL(t_u, t_e)	$t_u = 0$	Cross-sectional responder at t_e	$L = \int_{0}^{t_e} f(z)dz = F(t_u)$
LNNORMAL(t_u, t_e)	$t_u \neq t_e$	Interval censored over the interval (t_u, t_e). Includes, as a limiting case cross-sectional responder and right-censored.	$L = \int_{t_u}^{t_e} f(z)dz = S(t_u) - S(t_e)$
LNNORMAL(t_u, t_e, t_α)	$t_u = t_e$	Left-truncated exact failure	$L = \frac{f(t_u)}{\int_{t_\alpha}^{\infty} f(z)dz} = \frac{f(t_u)}{S(t_\alpha)}$
LNNORMAL(t_u, t_e, t_α)	$t_u \neq t_e$	Left-truncated, interval censored failure	$L = \frac{S(t_u) - S(t_e)}{\int_{t_\alpha}^{\infty} f(z)dz} = \frac{S(t_u) - S(t_e)}{S(t_\alpha)}$
LNNORMAL(t_u, t_e, t_α, t_ω)	$t_u = t_e$	Left- and right-truncated, exact failure	$L = \frac{f(t_e)}{\int_{t_\alpha}^{t_\omega} f(z)dz} = \frac{f(t_e)}{S(t_\alpha) - S(t_\omega)}$
LNNORMAL(t_u, t_e, t_α, t_ω)	$t_u < t_e$ $t_\alpha \leq t_u$ $t_\omega \geq t_e$	Left- and right-truncated, interval censored failure	$L = \frac{S(t_u) - S(t_e)}{\int_{t_\alpha}^{t_\omega} f(z)dz} = \frac{S(t_u) - S(t_e)}{S(t_\alpha) - S(t_\omega)}$
LNNORMAL(t_u, t_e, t_α)	t_u=t_e=t_α	Hazard	$L == \frac{f(t_u)}{S(t_u)} = h(t_u)$
LNNORMAL(t_u, t_e, t_α, t_ω)	t_u=t_e=t_α	Right-truncated hazard	$L == \frac{f(t_u)}{S(t_u) - S(t_\omega)} = h(t_u)$

The HAZARD statement always provides a proportional hazards specification modeled directly on the hazard of the PDF. Usually, the specification is loglinear, so that the hazard for the *i*th observation including the covariate effects defined as $h_i(t_i|\mathbf{x}_i\boldsymbol{\beta}) = h(t_i)\exp(\mathbf{x}_i\boldsymbol{\beta})$, where $h(t)$ is the baseline hazard for the specified PDF, and $\mathbf{x}_i\boldsymbol{\beta}$ is a vector of covariates $\boldsymbol{x_i}$ and parameters $\boldsymbol{\beta}$, so that $\mathbf{x}_i\boldsymbol{\beta} = x_{i1}\beta_1 + x_{i2}\beta_2 + x_{i3}\beta_3 \ldots$. Then, the survival function becomes $S_i(t_i|\mathbf{x}_i\boldsymbol{\beta}) = S(t_i)^{\exp(\mathbf{x}\boldsymbol{\beta})}$, and the probability density function becomes $f_i(t_i|\mathbf{x}_i\boldsymbol{\beta}) = f(t_i)S(t_i)^{\exp(\mathbf{x}\boldsymbol{\beta})-1}\exp(\mathbf{x}_i\boldsymbol{\beta})$.

This particular hazards model specification is commonly used. By exponentiating the $\mathbf{x}_i\boldsymbol{\beta}$ array, the covariate effects will never cause the hazard to go negative (hazards are *never* negative).

A multiplicative form for the proportional hazards specification can also be specified by setting the constant `EXP_HAZARD = FALSE` (it is `TRUE` by default). Then, the model is $h_i(t_i|\mathbf{x}_i\boldsymbol{\beta}) = h(t_i)\mathbf{x}_i\boldsymbol{\beta}$, $S(t_i|\mathbf{x}_i\boldsymbol{\beta}) = S(t_i)^{\mathbf{x}\boldsymbol{\beta}}$, and $f(t_i|\mathbf{x}_i\boldsymbol{\beta}) = f(t_i)S(t_i)^{\mathbf{x}\boldsymbol{\beta}-1}\mathbf{x}_i\boldsymbol{\beta}$. You must insure that $\mathbf{x}_i\boldsymbol{\beta}$ never goes negative.

The DATA function

The `DATA...END` function provides a mechanism to "feed" observations to the likelihood. This function specifies that observations are to be "fed" to the likelihood one at a time, corresponding to the product ($\prod$) over all observations shown in likelihoods (or the Σ shown in loglikelihoods). The `DATA` function loops through all observations that were previously read in by the `DATA` statement. In other words, the `DATA...END` function returns the *total logloglikelihood* or *total likelihood*, given a series of *observations* and an expression for an *individual likelihood* or *individual loglikelihood*. The general form for the `DATA` function is:

```
DATA <optional_form>
  <expression>
END
```

where optional_form is one of

- `FORM = SUMLL` — This takes the log of each individual likelihood and sums the loglikelihoods over the data. A likelihood (rather than a loglikelihood) is specified for <*expression*>. This is the default value if no <*formtype*> is specified.

- `FORM = SUM` or `FORM = SUMMATION` — Sums loglikelihoods over the data without first taking the log. This is used when a loglikelihood is specified rather than a likelihood for <*expression*>.

- `FORM = PROD` or `FORM = PRODUCT` — Takes the product of likelihoods over the data and does not take the log of the likelihood. This is used when a likelihood (rather than a loglikelihood) is specified for <*expression*> and some function appears outside the data function that takes the log.

  Here are three models that yield the same overall likelihood function, but uses different forms for the `DATA` function:

```
MODEL
  DATA  FORM = SUMLL        {the default form}
    PDF NORMAL(topen tclose)
      PARAM mu     LOW = 10     HIGH = 100 START = 30 END
      PARAM sigma  LOW = 0.0001 HIGH = 10  START = 1  END
    END  {pdf}
  END {data}
RUN
  FULL
END  {model}
```

```
MODEL
  DATA  FORM = SUM         {The loglikelihood is specified}
    LN(PDF NORMAL(topen tclose)
         PARAM mu     LOW = 10     HIGH = 100 START = 30 END
         PARAM sigma  LOW = 0.0001 HIGH = 10  START = 1  END
       END  {pdf}
    )
  END {data}
RUN
  FULL
END  {model}
```

```
MODEL
  LN(
    DATA  FORM = PRODUCT          {The likelihood is specified}
      PDF NORMAL(topen tclose)
        PARAM mu     LOW = 10     HIGH = 100 START = 30 END
        PARAM sigma  LOW = 0.0001 HIGH = 10  START = 1  END
      END  {pdf}
    END {data}
  )
RUN
  FULL
END  {model}
```

In theory, these three models will yield identical results. In practice, results may differ, particularly for the last model, because of round-off errors. The last likelihood will producd a very small number before the log is taken of the entire likelihood. There are several reasons for providing these three ways of specifying how the data is used within the likelihood:

- Some likelihoods are much easier to write as a loglikelihood.
- Many likelihoods include functions outside of the likelihood. In particular, some likelihoods take an expectation outside of the individual likelihoods so that integration is done outside of the data function.
- Some multilevel or hierarchical likelihoods require this type of control.

There are two functions that are closely related to the DATA function: the LEVEL function and the LEVELDELTA funciton. These two functions provides a mechanism by which multilevel or hierarchical models can be constructed.

The LEVEL function

The LEVEL function provides a mechanism by which multilevel or hierarchical models can be constructed. The syntax of the LEVEL function is

```
LEVEL <boolean expression> THEN <optional_form>
  <expression>
END
```

The effect of the LEVELDELTA function is to test the <*boolean expression*> for each observaton and, while the condition is true, form a product of likelihoods out of the observations. The <*optional_form*> is specified as it is for the DATA function, but with one difference: the default form is .FORM = PRODUCT.

The best way to understand the effect of the level command is by an example. Consider the likelihood

$$L = \prod_{i=1}^{N} \int_{\alpha}^{\omega} g(z) \left[\prod_{j=1}^{n_i} f(t_{i,j} \mid \theta, z) \right] dz \, .$$

This is a standard model for which a distribution of clustering (or heterogeneity), $g(z)$, is estimated along with the parameters ($\boldsymbol{\theta}$). There are two levels that make up this model. Let us call the outer level denoted by the outer product the *subject* level—that is, we have N subjects and this outer product is taken over those subjects. For each of N individuals, there are n_i observations. The inner level formed by the innermost product is the likelihood formed by n_i repeated observations of the ith subject.

The rationale for this type of model is that the repeated observations for individuals violate the condition that the likelihoods for each observation are independent. To fix this problem, we can compute an expected likelihood for each individual's observations. The integral computes the expected likelihood for each subject. Here is a concrete example

Say we have data in which levels are denoted by the number 1 or 2 as in

```
1  Tom Smith
2  23.4   26.8 . . .
2  19.2   22.9 . . .
2  26.8   -1   . . .
1  Steven Jones
2  19.5   23.7 . . .
2  26.8   -1   . . .
1  Martin Johnson
2  0      44.1 . . .
2  19.9   22.7 . . .
2  19.9   -1   . . .
...
```

where the observations beginning with a 2 correspond to the individual at the preceding 1, so that Tom Smith has three observations beginning 23.4, 19.2, and 26.8. If we were to treat all observations, within and among individuals, as independent, we could simply drop all of the level 1 lines, and form a likelihood as the product of all observations. But, if we want to treat observations within individuals as correlated (non-independent), the we can integrate over a distribution of common effects as shown in the likelihood above. Usually, we will estimate one or more parameters for the distribution $g(z)$, in addition to $\boldsymbol{\theta}$.

If we assume that $g(z)$ and $f(t)$ are normal distributions, the likelihood in ***mle*** would be specified as

```
MLE
  DATAFILE("example.dat")
  OUTFILE("example.out")

  DATA
    lev FIELD 1
    topen FIELD 2
    tclose FIELD 3
  END

  MODEL
    DATA
      LEVEL lev = 2 THEN
        INTEGRATE z (-12, 12)
          PDF NORMAL (z)
            0, PARAM sigmaz LOW = 0.0001  HIGH = 3 START = 0.2 END
          END {pdf}
         *
          PDF NORMAL(topen tclose)
            PARAM mu LOW = 10 HIGH = 100 START = 30 END
            PARAM sigma  LOW = 0.0001 HIGH = 10 START = 1 END
            HAZARD COVAR z 1
          END  {pdf}
        END {integrate}
      END {level}
    END {data}
  RUN
    FULL
  END  {model}
END  {mle program}
```

The `LEVEL` statement advances through all of the individual level observations and computes the product of the likelihoods for each individual. The `DATA` statement only "sees" observations that begin with a 1, because the `LEVEL` statement "consumes" all of the observations that begin with a 2. The `LEVEL` statement returns a likelihood, which is the product of likelihoods taken within each subject; the `DATA` statement takes those likelihoods, one per subject, takes the natural log of each, and sums them over all subject.

The LEVELDELTA function

The LEVELDELTA function is very similar to the LEVEL function. LEVELDELTA provides a mechanism by which multilevel or hierarchical models can be constructed. The syntax of the LEVELDELTA function is

```
LEVELDELTA <expression> THEN <optional_form>
  <expression>
END
```

The effect of the LEVELDELTA function is to evaluate <*expression*> for each observaton and, while the expression does not change, form a product of likelihoods out of the observations. The <*optional_form*> is specified as it is for the DATA function, but with one difference: the default form is .FORM = PRODUCT.

The only real difference between the LEVELDELTA and the LEVEL function is how each function decides when to "exit" the current level. The LEVELDELTA function simply looks for a change in the value of <*expression*> whereas LEVEL evaluates a boolean function <*bexpr*> for each observation and terminates when the expression evaluates to FALSE. In the example given under the LEVEL function, the only change necessary to use the LEVELDELTA function is replace the LEVEL line with

```
LEVELDELTA lev THEN
```

STATEMENTS AND PROCEDURES

Introduction

Previous chapters have described the MODEL statement and the DATA statement in some detail. A number of other statements are supported by ***mle*** as well. These include assignment statements, intrinsic procedures, which look like functions except that they do not return values, as well as looping statements (WHILE, REPEAT, FOR) and conditional statements (IF). This chapter summarizes all of the statements supported by ***mle***. In addition, a list of all procedures is given at the end of this chapter.

List of statements

Assignment statements

Assignment statements serve two purposes. First, assignment statements are used to define new variables. Secondly, assignment statements are used to assign a value to variables (whether new or pre-existing). The basic formats for assignment statements are

```
<var> = <expression>
<var>:<type> = <expression>
```

The <*var*> is the name of a variable. If the variable does not exist, it will be created. If so, and the first form of assignment is used, the type (INTEGER, REAL, etc.) returned from *expression* will define the type of the newly created variable. Under the second form of assignment, the variable type is specified by the <*type*>, where type is a type of variable. . The expression type must be compatible or an error will result. Some examples follow:

```
MLE
  y = SQRT(44.5)                {evaluates to 6.6708...}
  z = BETA(1.2, 9*3/10 + 1)     {evaluates to 0.185...}
  q = RAND                      {evaluates to a random number from 0 to 1}
  r:REAL = 2                    {defines r as real, assigns the value 2.0}
  ru : REAL                     {Defines ru, but does not initialize it}
  s = "This is a string"
  RANDOMSEED = 2342             {Set random number seed, or use the SEED(2342) procedure}
  b = IF RAND < 0.2 THEN TRUE ELSE FALSE END
  ra : REAL[-180 TO 180]        {Defines an array of type REAL with elements from -180 to 180}
  ia : INTEGER[0 TO 100, -1 TO 1] = 0  {Defines integer matrix and initialize elements to 0}
END
```

BEGIN...END statement

The BEGIN...END statement provides a means of providing multiple statements in contexts where only a single statement is allowed. The format is

```
BEGIN
  <statements>
END
```

The DATA statement

The DATA...END statement define the format of the data file, defines variables to be read in, and provides for some transformation of the variables. Details of the DATA statement is given in Chapter 3. Only an overview is given here. The format for the DATA statement is:

```
DATA
  <variable> [FIELD x [LINE y]] [=<expr>] [DROPIF <expr> | KEEPIF <expr> ...]
   ...
END
```

<*variable*> is the name of the variable to be defined. The variable must not already exist. All DATA variables are defined to be type real. Integer values will be read in and converted to real number values. Text strings can exist within a fields of a text file, but must not be assigned to a variable.

Field: The term *field* refers to which column within an input file a variable is found in. In the hammes.dat file used in Chapter 1, four fields (or columns) existed in the input file. The field specifier must be a positive integer constant.

Line: Sometimes observations are spread across multiple lines. When the LINE keyword is used, e.g. LINE 2, ***mle*** keeps track of the maximum number of lines specified this way. Then, *all* observations are assumed to have that number of lines. If the observations each take but one line, the statement LINE 1 may be dropped—one line per observation is assumed as a default. The line specifier must be a positive integer constant.

<=*expr*> defines a transformation expression. The expression can refer to the variable being read, or any variables that are defined before the current variable. The line `newvar FIELD 3 = newvar^2` will read newvar from field three of the data file. The value of newvar is then squared and assigned back to newvar.

<*DROPIF*> provides a mechanism to drop observations. The expression following DROPIF must evaluate to TRUE or FALSE. If TRUE, the expression is dropped. The line `newvar FIELD 3 DROPIF newvar <= 0` will drop all variables in field three that are not positive.

<*KEEPIF*> provides another mechanism to drop observations. The expression following KEEPIF must evaluate to TRUE or FALSE. If FALSE, the expression is dropped (that is, not kept). The line `newvar FIELD 3 KEEPIF newvar > 0` will drop all variables in field three that are not positive. KEEPIF and DROPIF expressions can be far more complex, but must return TRUE or FALSE.

Usually, data are read from a data file. The DATAFILE() procedure defines and opens this file. Thus

```
DATAFILE("test.dat")
DATA
  o_time      FIELD 1  = o_time*365.25
                         DROPIF (o_time > 100)
  c_time      FIELD 3  = IF c_time = -1 THEN c_time ELSE c_time*365.25 END
  missing     FIELD 4  DROPIF missing_data <> 1
  frequency   FIELD 5  DROPIF frequency <= 0
END
```

The variable names FREQUENCY or FREQ are taken as a field of frequencies for each observation. (If both variable names are used, FREQUENCY is taken as the frequency variable). The frequency of each observation is used to compute a proper likelihood. If the FREQUENCY or FREQ keywords are missing, one is assumed.

FOR statement

The FOR statement provides a means of looping through statements. The format is

```
FOR <v> = <expr> TO <expr> DO
  <statements>
END
```

and

```
FOR <v> = <expr> DOWNTO <expr> DO
  <statements>
END
```

The variable <*v*> must either not be previously defined or, if it already exists, it must be an integer variable. Its value will change as the FOR statement is executed. The first <*expr*> will be executed once and will define the starting value of *v*. The second <*expr*> will be executed once and will define the last value of *v*. Here is an example that will print sine and cosine tables in one degree increments as well as creating a table of radians for each degree:

```
r : REAL[0 TO 359]
FOR x = 0 TO 359 DO
  r[x] = DTOR(x)
  WRITELN(x " degrees (" r[x] " radians): SIN()=" SIN(r[x]) ", COS()=" COS(r[x]))
END
```

IF statement

The IF statement provides a means of conditionally executing statements. The following types of IF statements are available:

```
IF <bexpr> THEN
  <statements>
END
```

This form will conditionally execute the <*statements*> only if <*bexpr*> evaluates to TRUE. An ELSE clause can be added to the statement so that one of two sets of statements will always be executed:

```
IF <bexpr> THEN
  <statements>
ELSE
  <statements>
END
```

In addition, one or more ELSEIF clauses can be added to the statement to allow multiple conditions to be tested:

```
IF <bexpr> THEN
  <statements>
ELSEIF <bexpr> THEN
  <statements>
ELSEIF <bexpr> THEN
  <statements>
ELSE
  <statements>
END
```

Here is an example of using the IF statement:

```
IF SYSTEM = "MS-DOS" THEN
  PRINTLN("Run from an MS-DOS system")
  SEP = '\'
  DATAFILE("C:" + SEP + DIR + SEP + NAME)
ELSE
  PRINTLN("Run on a unix system")
  SEP = '/'
  DATAFILE(DIR + SEP + NAME)
END
```

MODEL statement

The MODEL...RUN...END statement defines the underlying probability model used by ***mle*** and also defines constraints on which parameters are to be estimated. An overview of the MODEL statement is given here. Chapter 4 gives details for the writing likelihood models using the MODEL statement.

The basic structure of the MODEL statement looks like this:

```
MODEL
  <expression>
RUN
  <run specifications>
END
```

Between MODEL and RUN is a single expression that is the likelihood. Within the likelihood is one or more PARAM...END functions. These define the parameters, whose values will be found so that the likelihood is maximized. One of the most important aspects of learning ***mle*** is the design and construction of the expression for the likelihood.

A list of <*run specifications*> is given between the RUN and the END part of the MODEL statement, this provides a way of evaluating the full model as well as a series of nested or reduced models. If all of the parameters (defined by PARAM...END functions) are to be found, a simple FULL command is placed between the RUN and its matching END. Reduced models, where one or more parameters are constrained to a constant or another parameter, are specified as REDUCE followed with a list of one or more "reductions". For example, you might constrain a parameter called mean to be zero and only allow the parameter called stdev to be found. Then you would put REDUCE mean = 0 between the RUN and the END. Any number of REDUCE commands (along with one FULL) can be used in a single model. The various forms of the model will be evaluated in turn.

Procedure statement

mle supports a number of intrinsic procedures statements. Procedures are single word commands that include zero or more arguments. Procedures perform some task given the list of arguments. Procedures do not return a value the way a function does. A list of all procedures, with examples, can be found at the end of this chapter.

REPEAT statement

The REPEAT statement provides a means of looping through statements until some condition is met. The format is

```
REPEAT
  <statements>
UNTIL <bexpr>
```

The <*statements*> are executed and then the boolean expression<*bexpr*> is evaluated. If the result is `FALSE`, the loop repeats and <*statements*> are executed again. When <*bexpr*> evaluates to `TRUE`, the loop terminates. A `REPEAT` statement always executes the <*statements*> at least once.

WHILE statement

The `WHILE` statement provides a means of looping through statements while some condition is met. The format is

```
WHILE <bexpr> DO
  <statements>
END
```

The boolean expression <*bexpr*> is executed first. If the value is `TRUE`, the <*statements*> are executed once and <*bexpr*> is evaluated again. The sequence continues until <*bexpr*> evaluates to `FALSE`. That is, when <*bexpr*> is `FALSE`, the loop terminates.

List of procedures

There are a few intrinsic procedures available in ***mle***. Procedures are single word commands that include zero or more arguments. Procedures perform some task given the list of arguments. Procedures do not return a value the way a function does. The following sections describes the procedures available in ***mle***.

DATAFILE(s)

Purpose:	Opens up a data file for the `DATA` statement. Closes any previous data files.
Arguments:	A single string expression
Examples:	`DATAFILE("mydata.dat")` `DATAFILE(filename + '.' + datextension)`
See also:	`OUTFILE`

HALT

Purpose:	Terminates execution of the program.
Arguments:	None

OUTFILE(s)

Purpose:	Opens up a standard output file to which results are printed. Closes any previous output files.
Arguments:	A single string expression
Examples:	`OUTFILE("mydata.out")` `OUTFILE(DEFAULTOUTNAME)`
See also:	`DATAFILE`, Function `DEFAULTOUTNAME`

PRINT(a1, a2, . . .)

Purpose:	Prints a message to the standard output file without including a carriage return at the end.
Arguments:	Any number of arguments of any type.
Examples:	`PRINT("The value of x is ", x)` `PRINT("Sin(x) squared is ", SIN(x)^2)`
See also:	`PRINTLN, OUTFILE, WRITE, WRITELN`

PRINTLN(a1, a2, . . .)

Purpose:	Prints a message to the standard output file and includes a carriage return at the end.
Arguments:	Any number of arguments of any type.
Examples:	`PRINTLN("The value of x is ", x)` `PRINTLN("Sin(x) squared is ", SIN(x)^2)`
See also:	`PRINT, OUTFILE, WRITE, WRITELN`

SEED(i)

Purpose:	Seeds the random number generator.
Arguments:	A single positive integer argument.
Examples:	`SEED(13234)` `SEED(x)`
See also:	functions `RAND`, `IRAND`, `RRAND`, variable `RANDOMSEED`

WRITE(a1, a2, . . .)

Purpose:	Writes a message to the terminal without including a carriage return at the end.
Arguments:	Any number of arguments of any type.

Examples:
```
WRITE("The value of x is ", x)
WRITE("Sin(x) squared is ", SIN(x)^2)
```

See also: WRITELN, OUTFILE, PRINT, PRINTLN, function PUT

WRITELN(a1, a2, . . .)

Purpose: Writes a message to the terminal and includes a carriage return at the end.

Arguments: Any number of arguments of any type.

Examples:
```
WRITELN("The value of x is ", x)
WRITELN("Sin(x) squared is ", SIN(x)^2)
```

See also: WRITE, OUTFILE, PRINT, PRINTLN, function PUT

FUNCTIONS

Introduction

This chapter describes all of the functions that are supported by ***mle***. Functions serve the purpose of returning a single value, be it a numeric value, a string value or a boolean value. Functions are used to build *expressions*, which are used for calculations of all types.

This chapter is something of a catalog of the functions provided by ***mle***. used All functions are listed in alphebetical order. At the end of this chapter is a list of all, so called, simple functions.

The DERIVATIVE function

The `DERIVATIVE` function computes the numerical value of the derivative at a particular point. Formats are:

```
DERIVATIVE <variable> = <exp1>, <exp2> END
DERIVATIVE <variable> = <exp1>, <exp2>, <exp3> END
DERIVATIVE (<expr4>) <variable> = <exp1>, <exp2> END
DERIVATIVE (<expr4>) <variable> = <exp1>, <exp2>, <exp3> END
```

The <*variable*> is the variable of differentiation. The first expression evaluates to the point at which the derivative is evaluated. The second expression is what will be differentiated. The optional third expression is the largest value of *dx* to begin with. If the third expression is not given, an initial value for *dx* of 0.001 is used, which is reasonable for a wide range of functions. This initial value can be changed by changing the value of the variable `DIFF_DX`. In finding the derivative, successively smaller values of *dx* are used until reasonable precision is reached.

For example, `DERIVATIVE x = 1, SIN(x) END` computes the value of the derivative at $x = 1$ for the function $\sin(x)$; it returns -0.841470984808. On Intel computers under DOS, this derivative requires about 4 function evaluations with an initial $dx = 0.001$. If *dx* is changed to 0.1, as in `DERIVATIVE x = 1, SIN(x), 0.1 END`, the same answer is found after 14 function evaluations. Likewise, if *dx* is set to 1E-7, 6 function evaluations are required.

The optional fourth expression evaluates to the degree of differentiation. If this value is not given or is ≤ 1, a first degree derivative will be found [i.e. $f'(x)$]. Higher degrees derivatives are found if the forth expression is specified and are greater than one. For example, SIN(x)" evaluated at $x = 1$ is equal to -SIN(1) which is -0.841470984808. The expression `DERIVATIVE (2) x = 1, SIN(x) END` returns the same numerical result.

High order derivatives tend to lose numerical precision. This can be seen in the following series, which should all evaluate to 24.0:

```
DERIVATIVE (1) x = 1, 24*x   END returns 24.000000000000
DERIVATIVE (2) x = 1, 12*x^2 END returns 23.999999999998
DERIVATIVE (3) x = 1, 4*x^3  END returns 24.000000000455
DERIVATIVE (4) x = 1, x^4    END returns 24.000026132114
```

The derivative function finds numerical estimates at a point using an adaptive algorithm similar to that described by Ridders (1982).

The FINDMIN function

The `FINDMIN` function iteratively finds the argument value that minimizes a bounded function. Formats are:

```
FINDMIN <variable> (<exp1>, <exp2>) <expr> END
FINDMIN <variable> (<exp1>, <exp2>, <expr3>) <expr> END
FINDMIN <variable> (<exp1>, <exp2>, <expr3>, <expr4>) <expr> END
FINDMIN <variable> (<exp1>, <exp2>, <expr3>, <expr4>, <expr5>) <expr> END
```

The <*variable*> is the argument that is changed to find the function minimum. The expression <*expr*> is the function to be minimized. The first expression is the minimum boundary of the function. The second expression is the maximum boundary of the function. The optional third expression is the starting value of <*variable*> to try. The optional forth expression is the desired precision of the solution. If the forth expression is not given, its value will taken from the variable `FIND_EPS`. The fifth expression is the maximum number of iterations allowed in finding the solution. If the fifth expression is not given, the value will be taken from the variable `FIND_ITERS`.

For example, `FINDMIN x (0, 2*PI) COS(x) END` which returns 3.1415925395570 (π is an exact solution). A more precise solution (but one that takes longer to find) is found by `FINDMIN x (0, 2*PI, 1, 1E-15) COS(x) END` which returns 3.1415926535713.

Finding the argument that maximizes the function is done by simply negating <*expr*>.

The FINDZERO function

The `FINDZERO` function iteratively finds the argument for which a bounded function is zero. Formats are:

```
FINDZERO <variable> (<exp1>, <exp2>) <expr> END
FINDZERO <variable> (<exp1>, <exp2>, <expr3>) <expr> END
FINDZERO <variable> (<exp1>, <exp2>, <expr3>, <expr4>) <expr> END
```

The <*variable*> is the argument that is changed to find the zero value of the function. The expression <*expr*> is the function to find zero for. The first expression is the minimum boundary of the function. The second expression is the maximum boundary of the function. The optional third expression is the desired precision of the solution. If the third expression is not given, its value will taken from the variable `FIND_EPS`. The forth expression is the maximum number of iterations allowed in finding the solution. If the forth expression is not given, the value will be taken from the variable `FIND_ITERS`.

For example, `FINDZERO x (0, PI) COS(x) END` returns 1.5707963267949, which is the correct value of π/2.

This function works for well-behaved functions that have a single continuous zero within the bounds. For example, cos(*x*) goes to zero for four different *x* values in the interval [0, 4π]. `FINDZERO x (0, 4*PI) COS(x) END` returns 10.995574287564 (and may return other solutions depending on hardware and some variable values). Another pathological example is functions with no zero in the specified interval. For example, cos(*x*) has no value of zero in the interval [2π/3, 4π/3]. `FINDZERO x (2*PI/3, 4*PI/3) COS(x) END` returns the value 2π/3; this is the closest value to zero found.

Identifiers and expressions

A very simple type of function is a constant or a variable (or a previously declared parameter which looks like a variable). Together these are called identifiers. *mle* predefines a number of built-in identifiers. Some predefined variables allow you to change the behavior of *mle*. A later chapter discusses many of those variables. Some constants arise frequently in numerical work, and are predefined for convenience. Some of these constants are given in Table 7.

Algebraic, boolean and logical expressions

Algebraic expressions are expressions created using a series of special operators. Operators include algebraic symbols like `+`, `-`, `*`, `/`, `^`, a series of algebraic keywords for integer operations,. `DIV`, `MOD`, `SHL`, `SHR`. The right hand side of an assignment statement is an expression. Thus, the following right-hand-sides are valid expressions:

```
n = 2*3
n = (HOURS/60)^2
n = 12.5*first - 10*second
i = mask SHL 4
i = 23 DIV 4
```

Boolean expressions evaluate to either `TRUE` or `FALSE`. The operators for creating boolean expressions are `>`, `<`, `>=`, `<=`, `==`, `<>`, and boolean keywords, `AND`, `OR`, `XOR`, and `NOT` and some simple functions. These operators are used in the same

Table 7 Some predefined mathematical constants.

Constant name	Meaning	Value ≈
`ATOMICMASSU`	atomic mass unit, 1/16 the mass of oxygen	1.6605655×10^{-27} kg
`AVOGADROSN`	N_a, Avogadro's number; atoms or molecules in 1 mole	6.022045×10^{-23} $(g\times mol)^{-1}$
`BOHRMAGNETON`	Bohr's magneton. A spinning electron's magnetic moment	9.274078×10^{-24} $A\times m^2$
`BOHRRADIUS`	Bohr's radius of the smallest electron orbit.	0.52917706×10^{-10} m
`BOLTZMANNSC`	Boltzsmann's constant, $k = R/N_a$	1.380662×10^{-23} J/K
`DEGREESPERRADIAN`	$180/\pi$	57.295779513
`E`	*e*, base of the Napierian logarithm	2.71828183
`EULERSCONSTANT`	γ, Euler's constant	0.57721566
`GRAVITATIONALC`	Gravitational force magnitude between two masses	6.672×10^{-11} $N\times m^2/kg^2$
`LIGHTC`	Speed of light in a vacuum	2.99792458×10^{8} m/s
`LOG_10`	ln(10)	2.3025850930
`PI`	π, ratio of any circle's circumference to its diameter	3.14159265
`PLANCKINV2PI`	$\hbar = h/(2\pi)$. See Planck's constant	1.054588×10^{-34} J×s
`PLANCKSC`	*h*, Planck's constant relating frequency of radiation to a quantum of energy.	6.626176×10^{-34} J×s
`RADIANSPERDEGREE`	$\pi/180$	0.0174532925
`RYDBERGC`	Rydberg's constant relating the spectral lines of hydrogen	1.097373177×10^{7} m^{-1}
`UNIVERSALGASC`	*R*, the universal gas constant	8.31441 m^{-1} (J×mol)/K
Number limits		
`OO`	The largest representable real number	machine dependent
`INFINITY`	The largest representable real number	machine dependent
`NEGINFINITY`	The most negative representable real number	machine dependent
`MACHINE_EPSILON`	The floating point precision	machine dependent
`SQRT_EPSILON`	The square root of `MACHINE_EPSILON`	machine dependent
`MAXINT`	The largest representable integer	machine dependent

way as they are in many other programming languages.

```
b = a <> 42^2
b = (a <> 12) AND (a >= 0)
```

The difference between boolean and logical expressions is that boolean expressions work with the values `TRUE` and `FALSE` only, whereas logical expressions work with bits. For example, `NOT TRUE` is equal to `FALSE`; but `NOT 767` is equal to `-768`. How does this work? The number 767 is represented by the computer as the binary sequence `00000000000000000000001011111111`. The logical `NOT` operator flips all 1s to 0s and 0s to 1s, so that the number becomes `11111111111111111111110100000000`. The first (left most) bit denotes a negative value, so the value is –768. The logical `AND`, `OR`, and `XOR` functions act bit by bit, as well. Thus the binary values `2X101101 AND 2X111000` (which is the same as `45 AND 56`) evaluates to 40 (or `2X101000`). The `SHL` and `SHR` operators shift bits to the left and right. So, `2X000111 SHL 3` (i.e. `7 SHL 3`) evaluates to 56 (or `2X111000`).

You might be wondering how ***mle*** decides whether an operator is boolean or logical. The answer is simple: if both operands are boolean types, the operator will be boolean. If both operands are integers, the operator will be logical. If one operator is boolean and one is logical, an error results. For the expression `(x >= 4) OR (y <= 2)`, each of the expressions in parenthesis will evaluate to `TRUE` or `FALSE`, so that the `OR` will be a boolean operator.

Operator precedence

The built in operators in ***mle*** follow a more or less standard precedence. That is, an expression like `4+2*3` will evaluate `2*3` first and then add `4`. The precedence of operators are defined in Table 9. Higher precedence operators will always be evaluated before lower precedence operators

The IF function

The `IF...THEN...ELSE...END` function permits conditional tests. The simplest format for the `IF` function is

```
IF <boolean expression> THEN
  <expression>
ELSE
  <expression>
END
```

If the <*boolean expression*> must return a boolean type. If the function evaluates to true then the first <*expression*> will be evaluated and returned as the value of the `IF` function. If the boolean function returns false, the second <*expression*> will be evaluated and returned as the result of the `IF` function. The <*expression*> may be another `IF` function, so that multiple `IF...THEN...ELSE` statements may be nested:

```
IF <condition> THEN
  <expression>
ELSE
  IF <condition> THEN
    <expression>
  ELSE
    IF <condition> THEN
      <expression>
    ELSE
      <expression>
    END          {3rd if...else}
  END             {2nd if...else}
END                {1st if...else}
```

An alternative to the above is to use a series of `ELSEIF <boolean expression> THEN <expression>` with the basic `IF` statement. The above example can be written

```
IF <condition> THEN
  <expression>
ELSEIF <condition> THEN
  <expression>
ELSEIF <condition> THEN
  <expression>
ELSE
  <expression>
END
```

Here are some examples of `IF` functions used in assignment statements:

```
ind = IF (a^2 > 12) OR (a = 0) THEN a^2 ELSE 0 END
```

Table 8. Algebraic, boolean, and logical operators.

Operator	Function	Example	Equivalent function
-	uniary negation	-x	NEGATE(x)
+	uniary positive	+x	
^	power function	x^y	POWER(x, y)
*	multiply function	x*y	MULTIPLY(x, y)
/	divide function	x/y	DIVIDE(x, y)
DIV	integer divide function	x DIV y	IDIV(x, y)
MOD	integer modulo function	x MOD y	MODF(x, y)
AND	boolean and logical and function	x AND y	ANDF(x, y)
SHL	logical shift left function	x SHL y	SHIFTLEFT(x, y)
SHR	logical shift right function	x SHR y	SHIFTRIGHT(x, y)
+	addition	x + y	ADD(x, y)
-	subtraction	x - y	SUBTRACT(x, y)
OR	boolean and logical or function	x OR y	ORF(x, y)
XOR	boolean and logical xor function	x XOR y	XORF(x, y)
== or =	boolean “is equal” function	x == y	ISEQ(x, y)
<>	boolean “not equal” function	x <> y	ISNE(x, y)
<	boolean “less than” function	x < y	ISLT(x, y)
>	boolean “greater than” function	x > y	ISGT(x, y)
<=	boolean “less than or equal to” function	x <= y	ISLE(x, y)
>=	boolean “greater than or equal to” function	x >= y	ISGE(x, y)

Table 9. Operator precedence.

Operator(s)	Precedence	Category
- + not	high	Uniary operators
^		Exponent operator
* / div mod and shl shr		Multiplying operators
+ - or xor		Adding operators
= (or ==) <> < > <= >=	low	Relational operators

```
message = IF not result THEN
            "The result is not valid"
          ELSE
            "The result is valid"
          END {if}

status = IF height < 48 THEN
           -1
         ELSEIF (height >= 48) and (height <= 60) THEN
           0
         ELSE
           1
         END
```

The INTEGRATE function

The INTEGRATE function does one-dimensional numerical integration. The integration method can be changed, and the user is given control on precision with some methods. Typically, INTEGRATE is used to integrate a likelihood over some distribution of unmeasured heterogeneity or to renormalize an improper (degenerate) density function. Two formats of INTEGRATE are

```
INTEGRATE <variable name> ( <lower_limit_expression> ,  <upper_limit_expression> )
  <expression>
END
```

and

```
INTEGRATE <variable name> ( <lower_limit_expression> ,  <upper_limit_expression>,
    <tolerance_expression> )
  <expression>
END
```

The *variable name* is the name of a variable of integration, which can be referenced within the *expression*. Within the parentheses that follow the variable are two expressions: one for the lower limit, and one for the upper limit of integration. These expressions are evaluated once prior to integration; The resulting values are then constant during the integration operations. The <*expression*> is the integrand, and can be any legal expression (including, perhaps, more INTEGRATE functions). For example, consider a model in which observed exact times (t) to failure are distributed according to a Weibull PDF, $f(t)$. In addition we model the distribution of unmeasured heterogeneity, $g(z) \sim N(0, \sigma_z^2)$. Assume that the effect of z on $f(t)$ is loglinear on a, so that the first parameter of the Gompertz distribution is $a'=ae^z$. The likelihood for the ith observation is:

$$L = \prod_{i=1}^{N} \int_{-25}^{25} g(z \mid 0, \sigma_z) f(t_i \mid ae^z, b) dz$$

In this model, we would like to estimate the parameters *a*, *b*, and σ_z from a series of observations. The ***mle*** code to estimate this model is

```
MODEL
  DATA
    INTEGRATE z (-25, 25)
        PDF NORMAL(z)       {g(z): heterogeneity}
          0, PARAM s  LOW = 0.0001 HIGH = 5  START = 1 END
        END {pdf normal}
      *
        PDF GOMPERTZ(t)      {f(t): distribution of failures}
          PARAM a LOW=0.0 HIGH=0.9 START=0.073 FORM=LOGLIN
            COVAR z 1, PARAM  b  LOW = 0.0001  HIGH = 1.4  START = 0.5 END
          END {param a}
        END {pdf Gompertz}
    END   {integrate}
  END {data}
END  {model}
```

The `INTEGRATE` function is particularly useful for 1) estimating distributions of unmeasured heterogeneity as shown in the example; 2) estimating multilevel models for which one can integrate out the non-independence of observations 3) dealing with left or left-interval censoring as is described in the Examples chapter; and 3) computing survivorship in custom likelihood when a closed form is not available for the PDF.

mle currently provides four different methods for performing numerical integration. Each method has its strengths and weaknesses. It should be kept in mind that numerical integration is difficult and time consuming—particularly once integrals become nested. One useful trick for successful integration is to set the limits of integration as narrow as possible without "shutting out" non-zero areas of the function to be integrated. In the above example, the limits of integration were set to ±25, because even with *s* = 5, the area under the distribution outside these limits is ignorable.

Adaptive quadrature. This is the default method, and it can be defined by setting `INTEGRATE_METHOD = I_AQUAD`. The method is an eight point adaptive quadrature integration routine adapted from the routine QUANC8 (Forsythe et al. 1977). The method will recursively integrate the function until a specified precision is reached. Precision is defined by changing the `INTEGRATE_TOL` constant. By default `INTEGRATE_TOL = 0.000001`. This method works well for relatively smooth functions, and is probably the best general integration routine incorporated into ***mle***.

Simpson. This method uses Simpson's rule to evaluate integrals to a predefined tolerance, and is defined by setting `INTEGRATE_METHOD = I_SIMPSON`. The method is adapted from the routine QSIMP (Press et al. 1986). The function will be integrated until a predefined precision is reached or a maximum number of iterations are reached. Precision is defined by changing the `INTEGRATE_TOL` constant. By default `INTEGRATE_TOL = 0.000001`. The maximum number of iterations is set with the constant `INTEGRATE_N` and is 100 by default. This method is useful for smooth functions.

Closed trapazoidal. This method uses a brute force extended trapazoidal function to evaluate integrals. The function to be integrated must be evaluable at the limits of integration; otherwise the opened trapazoidal should be used. The method is defined by setting `INTEGRATE_METHOD = I_TRAP_CLOSED`. The extended trapazoidal rule will be evaluated at a predefined number of equally spaced points defined by `INTEGRATE_N` (the default is 100). The minimum allowable steps is eight. The method does not provide an error tolerance. Even so, the error is on the order of $\texttt{INTEGRATE\_N}^{-4}$. This brute-force method is useful for functions that are not smooth enough for adaptive quadrature or Simpson.

Open trapazoidal. This method is similar to the closed trapazoidal method, except that the function to be integrated is never evaluated at the limits of integration. The method is defined by setting `INTEGRATE_METHOD = I_TRAP_OPENED`.

Table 10. Likelihoods returned by `PDF` for one, two, three, and four time variables under different conditions.

	Example	When	Class	Resulting Likelihood
1	`LNNORMAL` (t_e)		Exact failure at t_e	$L = f(t_e)$
2	`LNNORMAL` (t_u, t_e)	t_u=t_e	Exact failure at t_u=t_e	$L = f(t_u) = f(t_e)$
3	`LNNORMAL` (t_u, t_e)	t_e=○○ $t_e < t_u$	Right censored or cross-sectional non-responder at t_u	$L = \int_{t_u}^{\infty} f(z)dz = S(t_u)$
4	`LNNORMAL` (t_u, t_e)	$t_u = 0$	Cross-sectional responder at t_e	$L = \int_{0}^{t_e} f(z)dz = F(t_u)$
5	`LNNORMAL` (t_u, t_e)	$t_u \neq t_e$	Interval censored over the interval (t_u, t_e). Includes, as a limiting case cross-sectional responder and right-censored.	$L = \int_{t_u}^{t_e} f(z)dz = S(t_u) - S(t_e)$
6	`LNNORMAL` (t_u, t_e, t_α)	$t_u = t_e$	Left-truncated exact failure	$L = \frac{f(t_u)}{\int_{t_\alpha}^{\infty} f(z)dz} = \frac{f(t_u)}{S(t_\alpha)}$
7	`LNNORMAL` (t_u, t_e, t_α)	$t_u \neq t_e$	Left-truncated, interval censored failure	$L = \frac{S(t_u) - S(t_e)}{\int_{t_\alpha}^{\infty} f(z)dz} = \frac{S(t_u) - S(t_e)}{S(t_\alpha)}$
8	`LNNORMAL` (t_u, t_e, t_α, t_ω)	$t_u = t_e$	Left- and right-truncated, exact failure	$L = \frac{f(t_e)}{\int_{t_\alpha}^{t_\omega} f(z)dz} = \frac{f(t_e)}{S(t_\alpha) - S(t_\omega)}$
9	`LNNORMAL` (t_u, t_e, t_α, t_ω)	$t_u < t_e$ $t_\alpha \leq t_u$ $t_\omega \geq t_e$	Left- and right-truncated, interval censored failure	$L = \frac{S(t_u) - S(t_e)}{\int_{t_\alpha}^{t_\omega} f(z)dz} = \frac{S(t_u) - S(t_e)}{S(t_\alpha) - S(t_\omega)}$
10	`LNNORMAL` (t_u, t_e, t_α)	t_u=t_e=t_α	Hazard	$L == \frac{f(t_u)}{S(t_u)} = h(t_u)$
11	`LNNORMAL` (t_u, t_e, t_α, t_ω)	t_u=t_e=t_α	Right-truncated hazard	$L == \frac{f(t_u)}{S(t_u) - S(t_\omega)} = h(t_u)$

The LEVEL function

The `LEVEL` function provides a mechanism by which multilevel or hierarchical models can be constructed. The syntax of the `LEVEL` function is

```
LEVEL <boolean expression> THEN
  <expression>
END
```

The effect of the level statement is to test <*boolean expression*> and, while the condition is true, form a product of likelihoods out of the observations. The best way to understand the effect of the level command is by an example. The likelihood

$$L = \prod_{i=1}^{N} \int_{\alpha}^{\omega} g(z) \left[\prod_{j=1}^{n_i} f(t_{i,j} \mid \theta, z) \right] dz$$

is a standard model for which a distribution of clustering (or heterogeneity), $g(z)$, is estimated along with the parameters (θ). There are two levels that make up this model. The outer level denoted by the outer product is the *subject* level—that is, we have N subjects and this outer product is over those subjects. For each of N individuals, there are n_i observations. The inner level formed by the innermost product is the likelihood formed by n_i repeated observations of the ith subject.

The rationale for this type of model is that the repeated observations for individuals violate the condition that the likelihoods for each observation are independent. To fix this problem, we can compute an expected likelihood for each individual's observations. The integral computes the expected likelihood for each subject. Here is a concrete example

Say we have data in which levels are denoted by the number 1 or 2 as in

```
1  Tom Smith
2  23.4   26.8 . . .
2  19.2   22.9 . . .
2  26.8   -1   . . .
1  Steven Jones
2  19.5   23.7 . . .
2  26.8   -1   . . .
1  Martin Johnson
2  0      44.1 . . .
2  19.9   22.7 . . .
2  19.9   -1   . . .
...
```

where the observations beginning with a 2 correspond to the individual at the preceding 1, so that Tom Smith has three observations beginning 23.4, 19.2, and 26.8. If we were to treat all observations, within and among individuals, as independent, we could simply drop all of the level 1 lines, and form a likelihood as the product of all observations. But, if we want to treat observations within individuals as correlated (non-independent), the we can integrate out a distribution of common effects. The likelihood in ***mle*** would be specified as

```
MLE
  DATAFILE("example.dat")
  OUTFILE("example.out")

  DATA
    lev FIELD 1
    topen FIELD 2
    tclose FIELD 3
  END

  MODEL
    DATA
      LEVEL lev = 2 THEN
        INTEGRATE z (-12, 12)
          PDF NORMAL (z)
            0, PARAM sigmaz LOW = 0.0001  HIGH = 2 START = 0.2 END
          END {pdf}
          *
            PDF NORMAL(topen tclose)
              PARAM mu LOW = 10 HIGH = 100 START = 30  FORM = LOGLIN
                COVAR z 1
              END  {param}
              PARAM sigma  LOW = 0.0001 HIGH = 10 START = 1 FORM = NUMBER END
            END  {pdf}
        END {integrate}
      END {level}
    END {data}
  RUN
    FULL
  END  {model}
END  {mle program}
```

The PARAM *function*

mle has a general method for defining all parameters to be used in a likelihood model.[7] The PARAM function defines a parameter and its characteristics. When models are "solved", *free parameters* are estimated by iteratively plugging new values in for those parameters until the values that maximize the likelihood are found. In other words, free parameters are values that are to be estimated by ***mle*** —they are the unknowns in likelihood models. If the parameter is not constrained to some fixed value in the RUN part of the model statement, ***mle*** will estimate the value of that parameter.

Covariate effects (and their associated parameters) may be modeled within the parameter statement, as well. Parameters are specified as

```
PARAM  x HIGH = <expr>.LOW = <expr> START = <expr> TEST = <expr>..END
```

or

```
PARAM  x HIGH = <expr>.LOW = <expr> START = <expr> TEST = <expr>
  COVAR <expr>  PARAM  z ... END
END  {param}
```

Here is an example of a likelihood hand-coded for an exponential PDF for exact failure times. PARAMs, built-in functions, and pre-defined parameters are all used in this likelihood:

```
MODEL
  DATA
    PARAM lambda LOW = 0 HIGH = 1 START = 0.23 END * EXP(-lambda * t)
  END
RUN
  FULL
END
```

[7] The word *parameter* is used in a very specific way, as defined in Chapter 1. Parameters are the quantities to be estimated in a likelihood model

Notice that `lambda` is first defined as a parameter, and thereafter is used as an ordinary variable. As ***mle*** iteratively seeks a solution, the value of `lambda` will be updated. As the likelihood itself is being computed, the `PARAM` function will simply return the current estimate of `lambda`.

Sometimes parameters are constrained for the purpose of hypothesis testing. They may be held constant, or fixed to the value of another parameter. These are called *fixed parameters*, and an estimate will not be found for them. ***mle*** provides the mechanism for fixed parameters primarily to reduce models from more complicated to simpler forms. For example, in a slope function, we may have reason to believe that the slope m is one. Perhaps this is because of the nature of the physical system we are modeling. We could first fit our collection of x values to the model with parameter m free, and secondly fit it with m held constant to 1. Statistical criteria can be used to determine whether m deviates from the value we expected it to be.

Typically, parameters are defined for the intrinsic parameters of a `PDF` function. For example, the normal PDF has two intrinsic parameters μ and σ. The first parameter specified in the parameter list will be treated as μ. The second will be

Table 11. Forms and transformations for parameters.

Form	Parameter (p'), covariates ($\mathbf{x}_i$), covariate parameters ($\boldsymbol{\beta}$), and the value returned by the `PARAM` function (p_i)	Notes
`NUMBER`	$p_i = p$'	Default when no `COVAR`s are modeled.
`ADD`	$p_i = p' + \mathbf{x}_i\boldsymbol{\beta}$	Must be used with care when the resultant parameter is constrained to positive values because p_i might take on negative values for some combinations of $\mathbf{x}_i\boldsymbol{\beta}$
`INVERT`	$p_i = 1/(p' + \mathbf{x}_i\boldsymbol{\beta})$	The denominator must not be zero.
`INVADD`	$p_i = 1/p' + \mathbf{x}_i\boldsymbol{\beta}$	p' must not be zero.
`INVMULTIPLY`	$p_i = \mathbf{x}_i\boldsymbol{\beta}/p'$	p' must not be zero.
`INVLOGLIN`	$p_i = \exp(\mathbf{x}_i\boldsymbol{\beta})/p'$	p' must not be zero.
`DIVIDE`	$p_i = p'/\mathbf{x}_i\boldsymbol{\beta}$	$\mathbf{x}_i\boldsymbol{\beta}$ must not be zero.
`POWER`	$p_i = p'^{\mathbf{x}_i\beta}$	
`POWEREXP`	$p_i = p'^{\exp(\mathbf{x}_i\beta)}$	
`EXPADD`	$p_i = \exp(p' + \mathbf{x}_i\boldsymbol{\beta}) = \exp(p')\exp(\mathbf{x}_i\boldsymbol{\beta})$	Constrains p_i to positive values for all p' and $\mathbf{x}_i\boldsymbol{\beta}$.
`MULTIPLY`	$p_i = p' \times \mathbf{x}_i\boldsymbol{\beta}$	A multiplicative specification.
`EXCESS`	$p_i = p'\exp(1 + \mathbf{x}_i\boldsymbol{\beta})$	
`LOGLIN`	$p_i = p'\exp(\mathbf{x}_i\boldsymbol{\beta})$	This is a common specification, especially for parameters that are interpreted as hazards. When p' is constrained positive, the p_i will also be positive. Like `EXPADD` but $p'_{expadd} = \exp(p'_{loglin})$. `LOGLIN` is the default specification whenever a `COVAR` is defined.
`LOGISTIC`	If `ALTERNATE_LOGISTIC` = `FALSE`, $p_i = 1/[1 + \exp(p' + \mathbf{x}_i\boldsymbol{\beta})]$. If `ALTERNATE_LOGISTIC = TRUE`, $p_i = \exp(p' + \mathbf{x}_i\boldsymbol{\beta})/[1 + \exp(p' + \mathbf{x}_i\boldsymbol{\beta})]$	Frequently used for parameters that are interpreted as probabilities because, for all values of $p' + \mathbf{x}_i\boldsymbol{\beta}$, p_i will be constrained from zero to one. The alternative forms are related to each other as $p'_{form1} = 1 - p'_{form2}$
`LOGIT`	$p_i = \ln[\exp(p' + \mathbf{x}_i\boldsymbol{\beta})/(1 + \exp(p' + \mathbf{x}_i\boldsymbol{\beta})]$	This specification is useful when p_i can take on any value from $-\infty$ to ∞ and $p' + \mathbf{x}_i\boldsymbol{\beta}$ is a probability.

treated as σ. How can you know the proper order for parameters? Generally location parameters appear first (and are usually denoted *a* in this manual), scale parameters are second and shape parameters are third. Even so, you can get a quick synopsis of each type of PDF by using the -h option from the command line, e.g.: `mle -h SHIFTWEIBULL`

Parameters are also used to model effects of covariates on other parameters. Here is an example in which two parameters, used in place of some fixed values of μ and σ for a normal distribution, are defined with two covariate parameters, each:

```
PDF NORMAL(topen tclose)
  PARAM mean  LOW = 100  HIGH = 400  START = 270  TEST = 0  FORM = LOGLIN
    COVAR  sex    PARAM b_sex_mu     LOW = -2  HIGH = 2  START = 0 END
    COVAR  weight PARAM b_weight_mu  LOW = -2  HIGH = 2  START = 0 END
  END
  PARAM stdev     LOW = 0.1  HIGH = 100 START = 20  FORM = LOGLIN
    COVAR  sex    PARAM b_sex_sig     LOW = -2  HIGH = 2  START = 0 END
    COVAR  weight PARAM b_weight_sig  LOW = -2  HIGH = 2  START = 0 END
  END
END
```

In this example, the first parameter of the normal distribution (μ) has two covariates and their corresponding parameters modeled on it. The exact specification of how covariates and their parameters are modeled depend on the FORM of the intrinsic parameter. In the example, the `FORM = LOGLIN` specifies that a log-linear specification is to be used. The log-linear specification is $\mu_i = \mu' \exp(\mathbf{x_i}\boldsymbol{\beta})$, where μ' is the estimated intrinsic parameter (`mean` in this case). Thus, for the *i*th observation, the μ parameter of the normal distribution will be constructed as: $\mu_i = \texttt{mean} \times \exp(\texttt{sex}_i \times \texttt{b\_sex} + \texttt{weight}_i \times \texttt{b\_weight})$. The second parameter, `stdev`, has the same two covariates modeled on it, but the parameter names are (and must be) different from the parameters modeled on `mean`.

For some forms, the parameter itself is transformed. For example, when a parameter is a probability the parameter can be defined as:

```
PARAM p  LOW = -999  HIGH = 999  START = 0  FORM = LOGISTIC END
```

The logistic transformation permits the parameter `p` to take on any value from negative infinity to infinity, but the resulting value passed used by the likelihood will be constrained to the range (0, 1). In other words, ***mle*** will estimate a parameter over the range –999 to 999, but before that parameter is used in computation, it will undergo a logistic transformation as $p = 1/[1 + \exp(p')]$, so that the value of *p* will be a probability. ***mle*** currently provides a limited number of specifications for how parameters and covariates are modeled (Table 4). Even so, this mechanism for modeling covariates on any parameter is extremely general and provides the basis for building unique and highly mechanistic (Box et al. 1978) or etiologic (Wood 1994) models.

Setting Parameter Information

Four characteristics may be set for each parameter in addition to the FORM. They are: 1) the highest possible value that can be tried for the parameter, 2) the lowest possible value that can be tried for the parameter, 3) the starting guess to help ***mle*** out from the start, and 4) a test value against which the parameter will be tested when standard errors are computed. In the previous example, the `mean` parameter was constrained to the range [100, 400] and the initial guess was 270.

Use care when setting the HIGH and LOW limits. Most importantly, limits must be constrained to valid ranges for the intrinsic parameter. Thus, for the MIX mixing proportion parameter (the first of the three parameters) then, `HIGH = 1` and

`LOW = 0`, should be defined as is appropriate for a probability—unless some `FORM` like `FORM = LOGISTIC` is used to constrain the resulting parameter to between 0 and 1. Sometimes it is useful to impose narrower limits, perhaps to avoid getting hung-up at a local maximum or to solve the model more quickly. Be careful, though. Limits that are too narrow may exclude the global maximum—after all, the best parameter estimates for a set of data are presumably unknown. Excessively narrow limits may cause problems when numerical derivatives for the variance-covariance matrix are computed, as well. Also, likelihood confidence intervals will bump up and stop at the limits you set.

The `TEST = xxx` part of a `PARAM` function provides a value against which the parameter will be tested (in some reports). In a sense, the `TEST` value is a null hypothesis value (h_0). The test performed is $t = (\hat{p} - h_o) / SE(\hat{p})$, where $\hat{p}$ is the maximum likelihood parameter estimate and $SE(\hat{p})$ is the standard error for the parameter estimate. The *t*-test is provided for convenience only. ***mle*** does not make use of the test in any way.

The PDF function

The purpose of the `PDF` function is to specify the component of a pre-defined probability density or distribution functions in ***mle***. Although the name is `PDF`, the `PDF` function can return either the probability density function or specified

Table 12. Brief summary of some types of functions in ***mle***.

function	Brief description
`PDF`	Specifies a pre-defined probability or cumulative density function. Returns the value of the density or distribution function as is appropriate to the arguments with which is was called.
`INTEGRATE`	Integrates a function and returns the value of the integral.
`IF THEN ELSE`	Evaluates a condition and returns the appropriate subfunction.
`PREASSIGN`	Computes a subfunction and assigns the result to a variable. Then it computes and returns the second subfunction.
`POSTASSIGN`	Computes the first subfunction. Then it computes the second subfunction, assigns the result to a variable, and then returns the value of the first subfunction.
`PRODUCT`	Iterates over some limits and takes the product of a subfunction.
`SUMMATION`	Iterates over some limits and sums a subfunction.
function calls	A number of ordinary mathematical functions. Example: `SQRT(x)` and `ADD(x, y)`.
quick conditionals	`ZEROIF`, `ONEIF`, `NEGONEIF`, `INFINITYIF`, `NEGINFINIF`, `NEGONEIF`: return common values conditional on simple tests.
constants, variables	Pre-defined and user defined variables can be used as functions.
`PARAM`	A parameter to be estimated.
`DATA`	The data function cycles through all observations
`LEVEL`, `LEVELDELTA`	Creates a nested or multilevel likelihood.

areas under the PDF curve including the cumulative and survival density functions, and even the hazard function. In addition, the PDF function can return areas or densities that are left and right truncated. The structure of the PDF function call is:

```
PDF <PDF name> ( <time variable1>, <time variable2>, ... )
  <intrinsic parameter 1>,
  <intrinsic parameter 2>,
  ...
  <optional HAZARD parameter>
END
```

The *name* following PDF is the name of the built-in distribution. A brief summary for each built-in distribution is given in Table 13. A more complete description of each distribution is given in the Appendix.

Time variable list is a list of the time arguments passed to the PDF. Most univariate PDFs can take from one to four 'time' arguments.[8] In fact, these four times describe a single observation in such a way as to incorporate defects in the observation process (right censoring, left truncation, right truncation, cross-sectional). A description of how the four arguments combine to specify a probability are given in the section that follows. Note that the time arguments can be any expression, so that time shifts and transformations can be incorporated in this list.

Intrinsic parameter list provides specifications for the PDF's intrinsic parameters. The order that the intrinsic parameters are specified is important; it corresponds to how the PDF is defined within ***mle***.. The PDFs chapter lists the order for intrinsic parameters; alternatively, the command line `mle -h` can be used to determine the proper argument order. Note that any expression can be used for an intrinsic parameter. That is, you do not need to use a PARAM function for the intrinsic parameters, although this is the most common use. Here is an example in which the location parameter is fixed to a constant for a shifted lognormal distribution:

```
PDF SHIFTLOGNORMAL ( tooth_eruption_age )
  9,  {shift the time back to conception}
  PARAM location LOW = 1 HIGH = 4 START = 2.5 END,
  PARAM scale LOW = 0.0001 HIGH = 3 START = 0.9 END
END
```

PDF time arguments

Most PDFs can have as few as one and as many as four time arguments specified. They are: t_u, the last observation time before an event; t_e, the first observed time after the event; t_α, the left truncation time for the observation or the PDF; and t_ω, the right truncation time for the observation or the PDF. Understanding how these four times act on the PDF statement is critical to creating the desired and proper likelihood.

PDFs contribute to likelihoods in a number of ways. In survival analysis, for example, the likelihood for an exact failure is given by the value of the PDF at the exact point of failure. For a right censored observation, the likelihood is given by summing up (integrating) all possible PDF values from the last observation time until the maximum possible time. The likelihood for a cross-sectional "responder" is the integral from zero to the time of first observation. Table 6 lists the likelihoods that result from the four time variables for different conditions. For example, when t_u=t_e or when only one time variable is specified, ***mle*** returns the density at t_u. This is the desired likelihood for an exact failure. Likelihoods for right and interval censored observations, with and without left and right truncation are given in Table 6.

[8] These are called *time* variables in the context of survival analysis; however, they may represent other measurements (length, dose, height, etc.).

The Hazard Parameter

For most parametric distributions (like the normal or lognormal distributions) the hazard function does not take on a simple or closed form. For this reason, most studies have modeled the covariates as acting on the failure time for these distributions. Nevertheless, there is no inherent reason why hazards models cannot be constructed using distributions without a closed form for the hazards functions. Most of the PDFs built into ***mle*** provide a general mechanism for covariates to be modeled as affecting the hazard of failure, rather than (or in addition to) affecting intrinsic parameters. Here is an example:

```
PDF NORMAL(topen tclose)
  PARAM mean   LOW = 100  HIGH = 400  START = 270  TEST = 0  FORM = LOGLIN END,
  PARAM stdev  LOW = 0.1  HIGH = 100 START = 20 END,
  HAZARD COVAR  sex     PARAM b_sex     LOW = -2  HIGH = 2  START = 0 END
         COVAR  weight  PARAM b_weight  LOW = -2  HIGH = 2  START = 0 END
  END {hazard}
END
```

The covariates `sex` and `weight` are modeled to effect on the hazard of failure. Parameters `b_sex` and `b_weight` provide estimates of the effect.

The `HAZARD` statement always provides a proportional hazards specification modeled directly on the hazard of the PDF. Usually, the specification is loglinear, so that the hazard for the *i*th observation including the covariate effects defined as $h_i(t_i|\mathbf{x}_i\boldsymbol{\beta}) = h(t_i)\exp(\mathbf{x}_i\boldsymbol{\beta})$, where $h(t)$ is the baseline hazard for the specified PDF. Then, the survival function becomes $S_i(t_i|\mathbf{x}_i\boldsymbol{\beta}) = S(t_i)^{\exp(\mathbf{x}\boldsymbol{\beta})}$, and the probability density function becomes $f_i(t_i|\mathbf{x}_i\boldsymbol{\beta}) = f(t_i)S(t_i)^{\exp(\mathbf{x}\boldsymbol{\beta})-1}\exp(\mathbf{x}_i\boldsymbol{\beta})$. The reason for exponentiating the $\mathbf{x}_i\boldsymbol{\beta}$ array is to prevent it from going negative (hazards are *always* be positive).

A multiplicative form for the proportional hazards specification can also be specified by setting the constant `EXP_HAZARD = FALSE` (it is `TRUE` by default). Then, the model is $h_i(t_i|\mathbf{x}_i\boldsymbol{\beta}) = h(t_i)\mathbf{x}_i\boldsymbol{\beta}$, $S(t_i|\mathbf{x}_i\boldsymbol{\beta}) = S(t_i)^{\mathbf{x}\boldsymbol{\beta}}$, and $f(t_i|\mathbf{x}_i\boldsymbol{\beta}) = f(t_i)S(t_i)^{\mathbf{x}\boldsymbol{\beta}-1}\mathbf{x}_i\boldsymbol{\beta}$. You must insure that $\mathbf{x}_i\boldsymbol{\beta}$ never goes negative.

Table 13. Pre-defined distributions that may be used within the PDF function.

Name	Comments	Parameters	Variables
`ARCSINE`	Arcsine distribution	none	t_u, t_e, t_α, t_ω
`ASYMPTOTIC-RANGE`	Asymptotic range distribution	a (location), b (scale)	t_u, t_e, t_α, t_ω
`BERNOULLI-TRIAL`	Bernoulli distribution	p (proportion)	*outcome = 0 or non-0*
`BETA`	Beta distribution	a (shape), b (shape)	t_u, t_e, t_α, t_ω
`BINOMIAL`	Binomial distribution	p (probability), n (count)	t_u, t_e, t_α, t_ω
`BIRNBAUM-SAUNDERS`	Birnbaum-Saunders distribution	a (location), b (scale)	t_u, t_e, t_α, t_ω
`BIVNORMAL`	Bivariate normal distribution.	μ_x, σ_x, μ_y, σ_y, ρ (correlation)	t_{ux}, t_{uy}, t_{ex}, t_{ey}, $t_{\alpha x}$, $t_{\alpha y}$, $t_{\omega x}$, $t_{\omega y}$
`CAUCHY`	Cauchy distribution	a (location), b (scale)	t_u, t_e, t_α, t_ω
`CHI`	Chi distribution	a (location), b (scale), c (shape)	t_u, t_e, t_α, t_ω
`CHISQARED`	Chi squared distribution	a (location), b (scale)	t_u, t_e, t_α, t_ω
`COMPOUND-EXTREME`	Compound extreme	a (location), b (scale), c (shape)	t_u, t_e, t_α, t_ω
`DANIELS`	Daniel's distribution	none	t_u, t_e, t_α, t_ω
`DISK`	Disk distribution	a (location), b (scale)	t_u, t_e, t_α, t_ω
`EXPONENTIAL`	Exponential distribution	λ (hazard)	t_u, t_e, t_α, t_ω
`FAILED`	Returns 1 if a failure occurs or 0 if no failure.	none	t_u, t_e, t_α, t_ω

GAMMA	Gamma distribution	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
GAMMAFRAIL	Constant hazard with Gamma distributed heterogeneity	*h* (constant hazard), *c* (frailty distribution parameter)	t_u, t_e, t_α, t_ω
GAUSSIAN	Normal distribution. (NORMAL).	μ (location, mean), σ (scale, standard deviation)	t_u, t_e, t_α, t_ω
GENGAMMA	Generalized gamma distribution	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
GENGUMBEL	Generalized Gumbel distribution	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
GEOMETRIC	Geometric distribution	*p* (probability)	t_u, t_e, t_α, t_ω
GOMPERTZ	Gompertz PDF.	*a* (baseline hazard), *b* (time-dependent hazard)	t_u, t_e, t_α, t_ω
GUMBEL	Gumbel distribution. (LARGEEXTREME).	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
HORSESHOE	Horseshoe family including symmetric quad, quart, and sextic distributions	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
HYPERBOLIC-SECANT	Hyperbolicsecant distribution	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
HYPER-GEOMETRIC	Hypergeometric distribution	*p* (proportion), *m* (count) *n* (count)	t_u, t_e, t_α, t_ω
HYPER2EXP	Two point hyperexponential distribution	*p* (proportion); λ_1, λ_2 (subgroup hazards).	t_u, t_e, t_α, t_ω
HYPO2EXP	Two stage hypoexponential distribution	λ_1, λ_2 (subgroup hazards)	t_u, t_e, t_α, t_ω
IMMUNE	Returns 0 if failure occurs or 1 if no failure. Used with MIX to model sterility. (STERILE).	none	t_u, t_e, t_α, t_ω
INVBETA1	Inverse beta distribution. First type	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
INVBETA2	Inverse beta distribution. Second type	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
INVCHI	Inverse chi distribution.	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
INVGAMMA	Inverse gamma distribution.	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
INVGAUSSIAN	Inverse Gaussian distribution.	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
LAPLACE	Laplace distribution. Also called double, 2-tailed or bilateral exponential.	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
LARGEEXTREME	Largest extreme value distribution (type 1). (GUMBEL).	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
LINEARHAZARD	Linear hazard distribution	*a* (baseline), *b* (time-dependent)	t_u, t_e, t_α, t_ω
LNGAMMA	loggamma distribution	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
LNLOGISTIC	Two parameter log-logistic distribution.	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
LNNORMAL	Lognormal distribution. (LOGNORMAL).	*a* (location, median), *b* (scale)	t_u, t_e, t_α, t_ω
LOGISTIC	Two parameter logistic distribution	*a* (location), *b*(scale)	t_u, t_e, t_α, t_ω
LOGNORMAL	Lognormal distribution. (LNNORMAL).	*a* (location, median), *b* (scale)	t_u, t_e, t_α, t_ω
LOGSERIES	Logseries distribution.	*p* (proportion)	t_u, t_e, t_α, t_ω
LOWMAX	Lowmax distribution.	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
MAKEHAM	Gompertz-Makeham PDF	a_1, a_2, *b* (hazards)	t_u, t_e, t_α, t_ω
MAXWELL	Maxwell distribution.	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
MIXMAKEHAM	2-point mixed Gompertz-Makeham distribution.	*p* (proportion) a_1 (first constant hazard), a_2 (second constant hazard), a_2 , *b* (hazards).	t_u, t_e, t_α, t_ω
NEGBINOMIAL	Negative binomial distribution	*p* (proportion) *n* (count).	t_u, t_e, t_α, t_ω
NORMAL	The normal distribution. (GAUSSIAN)	μ (location, mean), σ (scale, standard deviation)	t_u, t_e, t_α, t_ω
PARETO	Pareto distribution.	*a* (location), *c* (shape)	t_u, t_e, t_α, t_ω
PASCAL	Pascal distribution.	*p* (proportion) *n* (count).	t_u, t_e, t_α, t_ω
POISSON	Poisson distribution.	*n* (count)	t_u, t_e, t_α, t_ω
POWER-FUNCTION	Power function distribution.	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
RASIED-COSINE	Raised cosine distribution.	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
RANDOMWALK	Random walk distribution.	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
RAYLEIGH	Rayleigh distribution.	*b* (scale)	t_u, t_e, t_α, t_ω
RECTANGULAR	Continuous uniform distribution. (UNIFORM)	none	t_u, t_e, t_α, t_ω
REVPOWER-	Reverse power function distribution.	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω

FUNCTION			
RINGINGEXP0	Ringing exponential distribution at 0 degrees phase.	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
RINGING-EXP180	Ringing exponential distribution at 180 degrees phase.	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
SHIFTED-LOGNORMAL	Shifted lognormal distribution.	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
SHIFTED-WEIBULL	Shifted weibull distribution.	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
SILER	Siler competing hazards distribution.	a_1, b_1, a_2, a_3, b_3 where, *a*s are baseline hazards, *b*s are time-dependent.	t_u, t_e, t_α, t_ω
SMALLEXTREME	Smallest extreme value distribution.	*a* (location), *b* (scale)	t_u, t_e, t_α, t_ω
SUBBOTIN	Subbotin distribution	*a* (location), *b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω
UNIFORM	Continuous uniform distribution. (RECTANGULAR)	none	t_u, t_e, t_α, t_ω
VONMESIS	Von Mesis distribution		t_u, t_e, t_α, t_ω
WEIBULL	Weibull distribution.	*b* (scale), *c* (shape)	t_u, t_e, t_α, t_ω

The PREASSIGN and POSTASSIGN functions

The likelihood is always specified as a single function. This means that within a likelihood, a special function must be used to compute intermediate results or perform other computations. The PREASSIGN. . . , . . .END function provides a mechanism to compute partial results of a likelihood outside of the main likelihood, or within part of the likelihood. The statement takes on this form

```
PREASSIGN
  <variable1> = <expression1>
  <variable2> = <expression2>
   ...
,
<expression>
END
```

One or more assignment statements are defined immediately after the PREASSIGN. Each of these assignment statements will be executed and the result assigned to the <*variable*> specified. After that, the <*expression*> (just before the END) is evaluated. The PREASSIGN function itself returns the results of that evaluation. For example, the following code would reparameterize the exponential PDF so that the parameter λ is replaced by the function $-b^{-1}$

```
MODEL
  PREASSIGN
    z = -1/PARAM b LOW = 0 HIGH = 1 START = 0.1 END
  ,
    DATA
      PDF EXPONENTIAL( t ) z END
    END {data}
  END {preassign}
RUN
  FULL
END
```

Notice that first the value z is assigned the value $-b^{-1}$. Next the likelihood $e^{\lambda t}$ is computed. But, lambda is constrained to *z*. The assignment in the first part of the PREASSIGN function will be executed for each observation.

The PREASSIGN function is particularly useful for defining a series of parameters, "up front", so that the likelihood function itself is easier to specify. Here is an example of recodeing a program so that all parameters are defined in advance.

```
MODEL          {mixture of two normal distributions}
  PREASSIGN
    pr = PARAM  p      LOW = 0    HIGH = 1   START = 0.5 END
    u1 = PARAM  mu1    LOW = 5    HIGH = 14  START = 8   END
    s1 = PARAM  sigma1 LOW = 0.1  HIGH = 5   START = 1.2 END
    u2 = PARAM  mu2    LOW = 0    HIGH = 6   START = 2   END
    s2 = PARAM  sigma2 LOW = 0.01 HIGH = 5   START = 1.2 END
  ,
    DATA
      MIX(pr, PDF NORMAL(topen tclose) u1 s1 END, PDF NORMAL(topcn tclose) u2 s2 END)
    END {data}
  END {preassign}
RUN
  FULL
END    {model}
```

The POSTASSIGN...,...END function is similar to the preassign function, except that the list of assignment statements come *after* the function expression. The form is

```
POSTASSIGN
  <expression>
,
  <variable1> = <expression1>
  <variable2> = <expression2>
  ...
END
```

The function defined by the first <*expression*> is evaluated first, and is the result returned by the POSTASSIGN function. Then the list of statements is evaluated with each result assigned to the corresponding <*variable*>.

The PRODUCT function

The PRODUCT function computes a finite product. The format is similar to the INTEGRATE or SUMMATION functions:

```
PRODUCT <variable name> (<lower_limit> , <upper_limit>)
  <expression>
END
```

or

```
PRODUCT <variable name> (<lower_limit> , <upper_limit>, <convergence>)
  <expression>
END
```

The expressions <*lower_limit*> and <*upper_limit*> define the lower and upper limits of the product. These expressions (as well as the optional <*convergence*> expression) are evaluated once. The optional <*convergence*> expression provides a second way to terminate the series. When used, the product will terminate when the difference between one product and the next is less than the value of the <*convergence*> expression

The PRODUCT function can be used, for example, to calculate likelihoods that incorporate geometric series. The inner <*expression*> will be repeatedly evaluated with the index variable incremented for each evaluation. Here is an example of a likelihood consisting of a Polya-Eggenberger distribution (Eggenberger and Pólya 1923) for exact failures at integer times *t*. The probability density function for the Polya-Eggenberger distribution is

$$f(t_i \mid p,n,c) = \frac{\binom{n}{t_i}\prod_{i=0}^{t_i-1}(p+ic)\prod_{j=0}^{n-t_i-1}(1-p+jc)}{\prod_{k=0}^{n-1}(1+kc)}$$

There are three products that must be computed as part of computing the density function. The following ***mle*** program fragment shows the code needed to implement the Polya-Eggenberger distribution. Parameters *p* and *c* are to be estimated for a set of observations **t**.

```
MODEL
  DATA
      COMBINATIONS(n, t)
    * PRODUCT i (0, t - 1)
          PARAM p LOW = 0 HIGH = 1 START = 0.5 END
        * i * PARAM c LOW = -1 HIGH = 25 START = 1 END
      END {product}
    *
      PRODUCT j (0, n - t - 1)
        1 - p + j*c
      END {product}
    /
      PRODUCT k (0, n - 1)
        1 - k*c
      END
  END {data}
RUN
  FULL
END
```

Simple functions

An <*expression*> can be a built-in simple function call, like `SIN( <expression> )`, `POWER( <expression> , <expression> )`, etc.. Simple functions are "simple" because they have a standard calling format. Given a list of zero or more arguments, they evaluate a function and return a single value. The function arguments themselves are <*expressions*>. Some examples are

```
x = FACT(4)                  {evaluates to 24}
y = SQRT(44.5)               {evaluates to 6.6708...}
z = BETA(1.2, 9*3/10 + 1)    {evaluates to 0.185...}
q = RAND                     {evaluates to a random number from 0 to 1}
```

A list all of the built-in simple functions comes at the end of this chapter.

The SUMMATION function

The `SUMMATION` function computes a finite sum. The format is similar to the `INTEGRATE` or `PRODUCT` functions:

```
SUMMATION <variable name> (<lower_limit> , <upper_limit>)
  <expression>
END
```

or

```
SUMMATION <variable name> (<lower_limit> , <upper_limit>, <convergence>)
  <expression>
END
```

The expressions <*lower_limit*> and <*upper_limit*> define the lower and upper limits of the sum. These expressions (as well as the optional <*convergence*> expression) are evaluated once. The optional <*convergence*> expression provides a second way to terminate the series. When used, the summation will terminate when the difference between one sum and the next is less than the value of the <*convergence*> expression

Here is an example of a likelihood hand-coded for a Thomas distribution (Thomas 1949) for exact failures at integer times *t*. The probability density function for the Thomas distribution is

$$f(t_i \mid a,b) = e^{-a} \sum_{i=1}^{t_i} \frac{e^{-ib} a^i (ib)^{t_i - i}}{i!(t-i)!}$$

A single finite summation must be computed as part of computing the density function. The following **mle** program fragment shows the code needed to implement the Thomas distribution. Parameters *a* and *b* are to be estimated for observed times *t*.

```
MODEL
  DATA
      EXP(-PARAM a LOW = 0.0001 HIGH = 20 START = 5 END)
    * SUMMATION i (1, t)
          EXP(-i * PARAM b LOW=0.0001 HIGH=40 START=0.5 END)
        * a^i * (i*b)^(t - 1) / (FACT(i)*FACT(t - 1))
      END  {summation}
  END      {data}
RUN
  FULL
END
```

List of simple functions

ABS(x)

Returns:	Absolute value of *x*
Range:	Positive values
Examples:	ABS(-4) returns 4
	ABS(4) returns 4
	ABS(-4.0) returns -4.0

ADD(x, y)

Returns:	The sum of two numbers, or the concatenation of strings or characters. This is the functional form of "*x* + *y*".
Examples:	ADD(1, 5) returns 6. The result is an integer.
	ADD(2.5, 2.5) returns 5.0. The result is a real number.
	ADD('a', " string") returns "a string".
Notes:	An integer is returned if both arguments are integers. A real value is returned if either argument is real. A string is returned with string or character arguments.

ANDF(x, y)

Returns:	Logical or boolean AND function. This is the functional from of *x* AND *y*.
Examples:	ANDF(TRUE, TRUE) returns TRUE
	ANDF(15, 28) returns 12
	ANDF(2x010101, 2x000111) returns 5
Notes:	If both *x* and *y* are integer types, ANDF(*x*, *y*) returns the bitwise (logical) AND of the two numbers. If *x* and *y* are boolean types, ANDF(*x*, *y*) returns the boolean AND of the two numbers.
See also	ORF, NOTF, XORF

ARCCOS(x)

Returns:	Inverse cosine of *x*, which is the angle (in radians) whose cosine is *x*
Constraints:	$-1 \le x \le 1$
Examples:	ARCCOS(0.5) returns 1.0.
	ARCCOS(-1/2) returns 2.0943951023932

ARCCOSH(x)

Returns:	Inverse hyperbolic cosine of *x*.
Constraints:	$x \ge 1$
Examples:	ARCCOSH(2) returns 1.3169578969248
	ARCCOSH(1) returns 0.0

ARCCOT(x)

Returns:	Inverse cotangent of *x*.
Constraints:	$x \ne 0$
Examples:	ARCCOT(3) returns 0.3217505543966

ARCCOTH(x)

Returns:	Inverse hyperbolic cotangent of *x*.
Constraints:	$x \ne 0$
Examples:	ARCCOTH(2) returns 0.5493061443341

ARCCSC(x)

Returns:	Inverse cosecant of *x*.
Constraints:	$x \neq 0$
Examples:	ARCCSC(5) returns 0.2013579207903

ARCCSCH(x)

Returns:	Inverse hyperbolic cosecant of *x*.
Constraints:	$x \neq 0$
Examples:	ARCCSCH(5) returns 0.1986901103492

ARCSEC(x)

Returns:	Inverse secant of *x*.
Constraints:	$x \neq 0$
Examples:	ARCSEC(1) returns 0.0

ARCSECH(x)

Returns:	Inverse hyperbolic secant of *x*.

ARCSIN(x)

Returns:	Inverse sine of *x*, or the number whose angle (in radians) is *x*
Constraints:	$-1 \leq x \leq 1$
Range:	$-\pi/2$ to $\pi/2$
Examples:	ARCSIN(1) returns 1.5707963267949
	ARCSIN(0.5) returns 0.5235987755983

ARCSINH(x)

Returns:	Inverse hyperbolic sine of *x*, which is the value whose hyperbolic sine is *x*
Examples:	ARCSINH(-2.5) returns -1.647231146371
	ARCSINH(0) returns 0.0

ARCTAN(x)

Returns:	Inverse tangent of *x*, which is the angle (in radians) whose tangent is *x*
Range:	$-\pi/2$ to $\pi/2$
Examples:	ARCTAN(0) returns 0.0
	ARCTAN(1) returns 0.7853981633974

ARCTANH(x)

Returns:	Inverse hyperbolic tangent of *x*.
Constraints:	$-1 < x < 1$
Examples:	ARCTANH(0) returns 0.0
	ARCTANH(0.5) returns 0.5493061443341

BESSELI(x, y)

Returns:	The modified Bessel function of the first kind *I* (integer order *x*) of real *y*.
See also:	BESSELJ, BESSELK, BESSELY

BESSELJ(x, y)

Returns:	The Bessel function of the first kind *J* (integer order *x*) of real *y*.
See also:	BESSELI, BESSELK, BESSELY

BESSELK(x, y)

Returns:	The modified Bessel function of the second kind *K* (integer order *x*) of real *y*.
See also:	BESSELI, BESSELJ, BESSELY

BESSELY(x, y)

Returns:	The Bessel function of the second kind *Y* (integer order *x*) of real *y*.
See also:	BESSELI, BESSELJ, BESSELK

BETA(ν, ω)

Returns: Euler's beta function. $\mathrm{BETA}(\nu,\omega)=\int_0^1 z^{\nu-1}(1-x)^{\omega-1}dx$.

Constraints: ν > 0, ω > 0

Examples: BETA(5, 2) returns 0.0095238095231

BETA(4.0, 8.0) returns 0.0007575757575

See also: BETA, IBETA, GAMMA, PDF BETA

BOOL2STR(x)

Returns: A string from boolean expression *x*.

Examples: BOOL2STR(1 <> 1) returns "FALSE"

BOOL2STR(TRUE) returns "TRUE"

See also: INT2STR, REAL2STR

CEIL(x)

Returns: The least integer greater than or equal to *x*.

Examples: CEIL(1.9) returns 2.0

CEIL(2.0) returns 2.0

CEIL(2.1) returns 3.0

CEIL(-1.9) returns -1.0

CEIL(-2.0) returns -2.0

CEIL(-2.1) returns -2.0

See also: FLOOR, ROUND, INT

COMB(x, y)

Returns: The binomial coefficient, which is combinations of *x1* elements taken *x2* at a time, which is: $x!/[y!\ (x-y)!]$

Constraints $x \geq y$, *x* and *y* are integer expressions.

Examples: COMB(13, 10) returns 286.0

COMB(5, 5) returns 1.0

See also: PERMUTATIONS

COMP(x)

Returns:	Complement of *x*. The results is computed as SIGN(1-ABS(*x*), *x*).
Examples:	COMP(0) returns 1.0
	COMP(0.25) returns 0.75
	COMP(1) returns 0
See also:	COMPN

COMPN(x, n)

Returns:	The *n* complement of *x*. The results is computed as SIGN(*n*-ABS(*x*), *x*).
Examples:	COMPN(4, 3) returns 1.0
	COMPN(-10, 2) returns -8.0
See also:	COMP

CONCAT(x1, x2)

Returns:	The concatenation of two strings or characters.
Examples:	CONCAT("hello", " world") returns "hello world"
	CONCAT('a', " string") returns "a string"
	CONCAT('a', 'b') returns "ab".
See also:	ADD

COS(x)

Returns:	Cosine of *x*.
Examples:	COS(0) returns 1
	COS(1) returns 0.5403023058681
	COS(DTOR(60)) returns 0.5

COSH(x)

Returns:	The hyperbolic cosine of *x*.
Examples:	COSH(0) returns 1
	COSH(4) returns 27.308232836016

COT(x)

Returns: Cotangent of *x*.

COTH(x)

Returns: The hyperbolic cotangent of *x*.

CSCH(x)

Returns: The hyperbolic cosecant of *x*.

DEC(x)

Returns: $x - 1$

Constraints: *x* must be an integer

Examples: DEC(42) returns 41

DEC(-42) returns -43

See also: INC

DEFALULTOUTNAME

Returns: A reasonable output file name based on the name of the ***mle*** program. The function appends ".out" to the program name after stripping off a trailing ".mle" or ".MLE", if any.

Examples: For programs called "sample.mle", "sample." or "sample" the function returns "sample.out"

Notes: The line `OUTFILE(DEFAULTOUTNAME)` will automatically pick and assign a useful name for the output file. This is useful when you are constantly modifying and changing the name of a program.

DELTA(x, y)

Returns: The Kronecker's delta function: 1 if $x = y$ otherwise 0.

Examples: DELTA(10, 10) returns 1

DELTA(11, 10) returns 0

SEE ALSO HEAVISIDE, SGN

DIVIDE(x, y)

Returns:	*x* divided by *y*. This is the functional form of *x*/*y*.
Constraints	$y \neq 0$
Examples	10/2 returns 5.0

DMSTOD(x, y, z)

Returns:	An angle in degrees from an angle in degrees (*x*), minutes (*y*), and seconds (*z*).
Examples:	DMSTOD(34, 15, 10.2) returns 34.252833333333
	DMSTOD(0, 20, 15) returns 0.33750

DMSTOR(x, y, z)

Returns:	An angle in radians from an angle in degrees (*x*), minutes (*y*), and seconds (*z*).
Examples:	DMSTOR(34, 15, 10.2) returns 0.5978247198035
	DMSTOR(0, 20, 15) returns 0.0058904862255

DMYTOJ(x, y, z)

Returns:	A Julian day from day (*x*), month (*y*), and year (*z*).
Examples:	DMYTOJ(15, 1, 2000) returns the Julian day 2451559
	DMYTOJ(4, 7, 1776) returns the Julian day 2369916
Notes:	This function is useful for computing durations between two dates in failure time models. For example, the duration between "birth" on 16 Feb 1976 and "death" on 21 Jul 1992 would be computed as DMYTOJ(21, 07, 1992) – DMYTOJ(16, 02, 1976), which returns exactly 6000 days.
See also:	JULIAND, JULIANM, JULIANY, YEARDAY, WEEKDAY

DTOR(x)

Returns:	Degrees from radians, $\pi x/180$.
Examples:	DTOR(30) returns 0.5235987755983
	DTOR(180) returns 3.1415926535898
See also:	RTOD

ERF(x)

Returns: The error function, which is the integral of a standard normal probability density function from 0 to *x*. $\mathrm{ERF}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2}\, du = 2\Phi\left(x\sqrt{2}\right) - 1$

Constraints: none, but *x* should be non-negative in order to return a proper probability

Examples: ERF(0.75) returns 0.7111555777692

ERF(2) returns 0.995322139784

ERF(-2) returns -0.995322139784, which is not a probability

See also: ERFC

ERFC(x)

Returns: The complementary error function, which is 1 – ERF(*x*).

Constraints: none, but *x* should be non-negative in order to return a proper probability

Examples: ERFC(0.75) returns 0.2888444222308

ERFC(2) returns 0.0046778602160

ERFC(-2) returns 1.9953221397840, which is not a probability

See also: ERF

EXP(x)

Returns: The value e raised to the power *x*, e^x.

Examples: EXP(0.2) returns 1.2214027581602

EXP(0) returns 1

EXP(-0.2) returns 0.1353352832366

FACT(x)

Returns: The factorial function, x!, which is $x \times (x–1) \times (x–2) \times \ldots \times 2 \times 1$.

Constraints: None, but the function can overflow the computers representation of a real number from modest values of *x*

Examples: FACT(5.0) returns 120.0

FACT(100) returns 9.332622E+0157

FISHER(x)

Returns:	Fisher's transformation as $\frac{1}{2}\ln\left(\frac{1+x}{1-x}\right)$.
Constraints	$-1 < x < 1$
Examples:	FISHER(0) returns 0.0
	FISHER(0.5) returns 0.5493061443341
	FISHER (-0.99990000) returns -4.951718775643
See also:	FISHERINV

FISHERINV(x)

Returns:	The inverse Fisher's transformation as $\frac{\exp(2x)-1}{\exp(2x)+1}$.
Range:	The result falls between -1 and 1.
Examples:	FISHERINV(4) returns 0.9993292997391
	FISHERINV(0.5) returns 0.4621171572600
	FISHERINV(0) returns 0.0
See also:	FISHER

FLOOR(x)

Returns	The greatest integer less than or equal to *x* as a real number.
Examples:	FLOOR(1.9) returns 1.0
	FLOOR(2.0) returns 2.0
	FLOOR(2.1) returns 2.0
	FLOOR(-1.9) returns -2.0
	FLOOR(-2.0) returns -2.0
	FLOOR(-2.1) returns -3.0
See also:	CEIL, ROUND, INT

FRAC(x)

Returns:	The fractional part of x.
Examples:	FRAC(2.0) returns 0.0
	FRAC(2.1) returns 0.1

FRAC(-3.2) returns -0.2

See also: INT

GAMMA(x)

Returns: Euler's gamma function, $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt$

Examples: GAMMA(4) returns 6

GAMMA(PI) returns 2.2880377950731

See also: IGAMMA, IGAMMAC, IGAMMAE, PDF GAMMA

GCF(x, y)

Returns: The greatest common factor of x and y, which is the greatest value that divides both x and y exactly.

Examples: GCF(81, 36) returns 9

GCF(143, 187) returns 11

See also: LCM

HEAVISIDE(x)

Returns: the Heaviside function, which is 1 if $x \geq 0$ otherwise it returns 0.

Examples: HEAVISIDE(3) returns 1

HEAVISIDE(0) returns 1

HEAVISIDE(-2) returns 0

See also: DELTA, SGN

IBETA(p, ν, ω)

Returns: The normalized Euler's incomplete beta function. $\mathrm{BETA}(p,\nu,\omega) = \dfrac{\int_0^p x^{\nu-1}(1-x)^{\omega-1}\, dx}{\mathrm{BETA}(\nu,\omega)}$.

Constraints: $0 \leq p \leq 1$, $\nu > 0$, $\omega > 0$

Examples IBETA(0.5, 3, 6) returns 0.8554687499873

IBETA(1, 3, 3) returns 1.0

See also: BETA, IBETAC, GAMMA, PDF BETA

IBETAC((p, ν, ω)

Returns: The complement of the normalized Euler's incomplete beta function. IBETAC(p, ν, ω) = 1 – IBETA(p, ν, ω).

Constraints: $0 \le p \le 1$, $\nu > 0$, $\omega > 0$

Examples: IBETAC(0.5, 3, 6) returns 0.1445312500127

IBETAC(1, 3, 3) returns 0.0

See also: BETA, IBETA, GAMMA, PDF BETA

IDIV(x, y)

Returns: The integer part of x/y. This is the same as the algebraic expression x DIV y.

Constraints: $y \neq 0$. x and y must be integers

Examples: IDIV(104, 25) returns 4

IDIV(-124, 25) returns -4

See also: MODF, MODULO, REMAINDER, DIVIDE

IGAMMA(x, y)

Returns: Euler's incomplete gamma function.

See also: GAMMA, IGAMMAC, IGAMMAE

IGAMMAC(x1, x2)

Returns: The complement of the Euler's incomplete gamma function.

See also: GAMMA, IGAMMA, IGAMMAE

IGAMMAE(x1, x2)

Returns: IGAMMA(x1, x2)*ROOT(x2, x1).

INC(x)

Returns:	$x + 1$
Constraints:	x must be an integer
Examples:	INC(42) returns 43
	INC(-42) returns -41
See also:	DEC

INT(x)

Returns	The integer part of x as a real number.
Examples:	INT(1.9) returns 1.0
	INT(2) returns 2.0
	INT(-1.9) returns -1.0
See also:	FRAC, CEIL, ROUND, FLOOR

INT2STR(x)

Returns:	A string from integer expression x.
Examples:	INT2STR(-123) returns "-123"
	INT2STR(0rMCMXII) returns "1912"
See also:	BOOL2STR, REAL2STR

INVERT(x)

Returns:	$1/x$.
Constraints	$x \neq 0$
Examples:	INVERT(2) returns 0.5
	INVERT(-1.25) returns -0.8

IRAND(x, y)

Returns:	A random integer from x to y.
Notes:	Before IRAND or other random number functions can be used, the value of RANDOMSEED must be set to a positive constant. Use the SEED() procedure.
See also:	RAND, RRAND

ISEQ(x, y)

Returns: The boolean $x = y$

See also: ISGE, ISGT, ISLE, ISLT, ISNE, ISNEAR

ISEVEN(x)

Returns: TRUE if integer x is an even number, FALSE if x is odd

Examples: ISEVEN(0) returns TRUE

ISEVEN(199) returns FALSE

See also: ISODD

ISGE(x, y)

Returns: The boolean $x \geq y$

See also: ISEQ, ISGT, ISLE, ISLT, ISNE, ISNEAR

ISGT(x, y)

Returns: The boolean $x > y$

See also: ISGE, ISEQ, ISLE, ISLT, ISNE, ISNEAR

ISLE(x, y)

Returns: The boolean $x \leq$ y

See also: ISGE, ISGT, ISEQ, ISLT, ISNE, ISNEAR

ISLT(x, y)

Returns: The boolean $x < y$

See also: ISGE, ISGT, ISLE, ISEQ, ISNE, ISNEAR

ISNE(x, y)

Returns: The boolean $x \neq y$.

See also: ISGE, ISGT, ISLE, ISLT, ISEQ, ISNEAR

ISNEAR(x, b, δ)

Returns: TRUE if x is in the interval $[b - \delta, b + \delta]$; otherwise it returns FALSE.

Examples: ISNEAR(23.5, 20, 4) returns TRUE

ISNEAR(23.5, 20, 1) returns FALSE

See also: ISGE, ISGT, ISLE, ISLT, ISNE, ISEQ

ISODD(x)

Returns: TRUE if integer x is an odd number, FALSE if x is even

Examples: ISODD(0) returns FALSE

ISODD(199) returns TRUE

See also: ISEVEN

JULIAND(x)

Returns: The day of the month for a Julian day.

Range: 1 to 31

Examples: DMYTOJ(30, 6, 1961) returns 2437481 {30 June 1961}

JULIAND(2437481) returns 30

See also: JULIANM, JULIANY, YEARDAY, WEEKDAY, DMYTOJ

JULIANM(x)

Returns: The month for a Julian day.

Range: 1 to 12

Examples: DMYTOJ(30, 6, 1961) returns 2437481 {30 June 1961}

JULIANM(2437481) returns 6

See also: JULIAND, JULIANY, YEARDAY, WEEKDAY, DMYTOJ

JULIANY(x)

Returns: The year for a Julian day.

Examples: DMYTOJ(30, 6, 1961) returns 2437481 {30 June 1961}
JULIANY(2437481) returns 1961
See also: JULIAND, JULIANM, YEARDAY, WEEKDAY, DMYTOJ

LCM(x, y)

Returns: The least common multiple of *x* and *y*.
Examples: LCM(9, 12) returns 36
See also: GCF

LEAPYEAR(y)

Returns: The TRUE if the year specified by *y* is a leap year.
Examples: LEAPYEAR(2000) returns TRUE
LEAPYEAR(2001) returns FALSE

LEFTSTRING(x, y)

Returns: The leftmost substring from *x* of up to *y* characters.
Examples: LEFTSTRING("Probability theory", 11) returns "Probability"
LEFTSTRING("Anyway", 3) returns "Any"
LEFTSTRING("Anyway", 20) returns "Anyway"
See also: RIGHTSTRING, SUBSTRING

LN(x)

Returns: The natural (Naperian) logarithm of *x* (also LOG).
Constraints $x \geq 0$. If $x = 0$, $-\infty$ is returned.
Examples: LN(E) returns 1.0
LN(E^25) returns 25
LN(35) returns 3.5553480614894
See also: LOG, LOGBASE, EXP

LNFACT(x)

Returns:	The natural logarithm of *x*!.
Example	LNFACT(10) returns 15.104412573076
	LNFACT (3000) returns 21024.024853046
See also:	FACT, LNGAMMA

LNGAMMA(x)

Returns:	The natural logarithm of GAMMA(*x*).
Example:	LNGAMMA (11) returns 15.104412572871
	LNGAMMA(3001) returns 21024.024853046
See also:	GAMMA, LNFACT

LOG(x)

Returns:	The natural (Naperian) logarithm of *x* (also LN).
Constraints	$x \geq 0$. If $x = 0$, $-\infty$ is returned.
Examples:	LOG(E) returns 1.0
	LOG(E^25) returns 25
	LOG(35) returns 3.5553480614894
See also:	LN, LOGBASE, LOG10, EXP

LOG10(x)

Returns:	The base-10 logarithm *x*.
Constraints	$x \geq 0$. If $x = 0$, $-\infty$ is returned.
Examples:	LOG10(10) returns 1.0
	LOG10(10^25) returns 25
	LOG10(35) returns 1.5440680443503
See also:	LN, LOG, LOGBASE, EXP

LOGBASE(x, y)

Returns:	The logarithm (base *y*) of *x*.
Constraints	$x \geq 0$. If $x = 0$, $-\infty$ is returned.

Examples: LOGBASE(10, 10) returns 1.0
LOGBASE(10^25, 10) returns 25
LOGBASE(35, E) returns 3.5553480614894
See also: LN, LOG, LOG10, EXP

LOGISTIC(x)

Returns: 1/[1 + exp(x)] (its complement if alt_logistic=true).

LOGIT(x)

Returns: ln[exp(x)/(exp(x) - 1)].

LUNARPHASE(j)

Returns: An approximate phase of the moon on Julian date *j*.
Examples: LUNARPHASE(DMYTOJ(15, 1, 2000)) returns 0.6055993482011
LUNARPHASE(DMYTOJ(16, 1, 2000)) returns 0.6733257524912
See also: DMYTOJ

MAX(x, y)

Returns: The greatest of *x* and *y*.
Examples: MAX(15, 10) returns 15
MAX(-15, -10) returns -10
See also: MIN

MIN(x, y)

Returns: The least of *x* and *y*.
Examples: MIN(10, 15) returns 10
MIN(-15, -10) returns -15

MIX(p, x, y)

Returns:	x and y weighted (mixed) by probability p. $px + (1 - p)y$.
Examples:	MIX(0.5, 10, -10) returns 0
	MIX(0.25, 1000, 2000) returns 1750

MODULO(x, y)

Returns:	The integer remainder of x/y. This is the same as the algebraic expression x MOD y.
Constraints	$y \neq 0$. x and y must be integers
Examples:	MODULO(110, 25) returns 10
	MODULO(-124, 25) returns -24
See also:	REMAINDER, IDIV

MONTHDAYS(m, y)

Returns:	the number of days in month m of year y.
Examples:	MONTHDAYS(2, 2000) returns 29
	MONTHDAYS(2, 2001) returns 28
See also:	LEAPYEAR

MULTIPLY(x, y)

Returns:	The algebraic product $x \times y$; This is the same as the algebraic expression x * y.
Examples:	MULTIPLY(2, 3) returns 5
	MULITPLY(2.5, 3) returns 7.5

NEGATE(x)

Returns:	$-x$.
Examples:	NEGATE(23) returns -23
	NEGATE(-45) returns 45

NOTF(x)

Returns:	The logical or boolean NOT function.
Examples:	NOTF(357) returns -358
	NOTF(2x011011) returns -28
	NOTF(TRUE) returns FALSE
Notes:	If *x* is an integer, NOTF(*x*) returns the bitwise (logical) NOT of the number. If *x* is a boolean variable or constant, NOTF(*x*) returns the boolean NOT of the number.
See also	ORF, ANDF

ORD(c)

Returns:	The ordinal value of character *c*.
Examples:	ORD('A') returns 65
	ORD('a') returns 97

ORF(x, y)

Returns:	The logical or boolean or function.
Examples:	ORF(456, 123) returns 507
	ORF(2x00101, 2x01010) returns 15
	ORF(TRUE, FALSE) returns TRUE
Notes:	If both *x* and *y* are integer variables or constants, ORF(*x*, *y*) returns the bitwise (logical) OR of the two numbers. If *x* and *y* are boolean variables or constants, ORF(*x*, *y*) returns the boolean AND of the two numbers.
See also:	NOTF, ANDF, XORF

PERMUTATIONS(x, y)

Returns:	Permutations: *x* taken *y* at a time: $x!/(x-y)!$.
Constraints	$x \geq y$, *x* and *y* are integer expressions.
Examples:	PERMUTATIONS(10, 1) returns 10.0
	PERMUTATIONS(10, 3) returns 720.0
See also:	COMB

POLARTORECTX(r, a)

Returns: Rectangular *x* coordinate from polar coordinates *r*, *a*, *r*cos(*a*).

See also: POLARTORECTY, RECTTOPOLARR, RECTTOPOLARA

POLARTORECTY(r, a)

Returns: Rectangular *y* coordinate from polar coordinates *r*, *a*: *r*sin(*a*).

See also: POLARTORECTX, RECTTOPOLARR, RECTTOPOLARA

POWER(x, y)

Returns: *x* raised to the power *y*, x^y. This is the functional form of the algebraic expression *x^y*.

See also: ROOT

PUT(x)

Returns: the value of *x*, and, as a side effect, writes *x* to the standard output.

Notes: *x* can be any type. This function is particularly useful for debugging likelihoods.

RAND

Returns: A random real number from 0 to 1.

Notes: Before RAND or other random number functions can be used, the value of RANDOMSEED must be set to a positive constant.

See also: IRAND, RRAND

REAL2STR(x, l, s)

Returns: A string from real expression *x*. The length of the string is *l* characters, and at least *s* significant digits are represented. The function tries to return the number in decimal format rather than scientific format. A minimum field length of about 9 is recommended for small or large numbers.

Examples: REAL2STR(PI, 10, 1) returns "3.14159265"

REAL2STR(PI, 5, 5) returns "3.142"

REAL2STR(1.234567E-8, 20, 2) returns "0.000000012345670000"

REAL2STR(1.234567E-8, 9, 8) returns "1.2E-0008"

REAL2STR(1.234567E-8, 12, 5) returns "1.235E-0008"

See also: BOOL2STR, INT2STR

RECTTOPOLARA(x, y)

Returns: Polar angle (in radians) from rectangular coordinates *x* and *y*.

See also: POLARTORECTY, POLARTORECTA, RECTTOPOLARR

RECTTOPOLARR(x, y)

Returns: Polar radius from rectangular coordinates *x* and *y*.

See also: POLARTORECTY, POLARTORECTA, RECTTOPOLARA

RECTTOSPHERER(x, y, z)

Returns: Spherical radius from rectangular coordinates *x*, *y*, and *z*.

Examples: RECTTOSPHERER(1, 2, 3) returns 3.7416573867739

RECTTOSPHERER(1, 1, 1) returns 1.7320508075689

See also: SPHERETORECTX, SPHERETORECTY, SPHERETORECTZ, RECTTOSPHEREA1, RECTTOSPHEREA2, RECTTOPOLARA, RECTTOPOLARR, POLARTORECTX, POLARTORECTY

RECTTOSPHEREA1(x, y, z)

Returns: Spherical angle 1 (in radians) from rectangular coordinates *x*, *y*, and *z*.

Examples: RECTTOSPHEREA1(1, 2, 3) returns 1.1071487177941

RECTTOSPHEREA1(1, 1, 1) returns 0.7853981633974

See also: SPHERETORECTX, SPHERETORECTY, SPHERETORECTZ, RECTTOSPHERER, RECTTOSPHEREA2, RECTTOPOLARA, RECTTOPOLARR, POLARTORECTX, POLARTORECTY

RECTTOSPHEREA2(x, y, z)

Returns: Spherical angle 2 (in radians) from rectangular coordinates *x*, *y*, and *z*.

Examples: RECTTOSPHEREA2(1, 2, 3) returns 0.6405223126794

RECTTOSPHEREA2(1, 1, 1) returns 0.9553166181245

See also: SPHERETORECTX, SPHERETORECTY, SPHERETORECTZ, RECTTOSPHERER, RECTTOSPHEREA1, RECTTOPOLARA, RECTTOPOLARR, POLARTORECTX, POLARTORECTY

REMAINDER(x, y)

Returns: The real remainder of x/y. Returns 0 if $y = 0$

Examples: REMAINDER(110.0, 25.0) returns 10.0

REMAINDER(-124, 25) returns -24.0

REMAINDER(-100, 0) returns 0.0

See also: MODULO, IDIV

RIGHTSTRING(x, y)

Returns: The rightmost substring from *x* of up to *y* characters.

Examples: RIGHTSTRING("Probability theory", 6) returns "theory"

RIGHTSTRING("Small", 4) returns "mall"

RIGHTSTRING("Small", 20) returns "small"

See also: LEFTSTRING, SUBSTRING

ROOT(x, y)

Returns: The *y*th root of *x*, $x^{1/y}$.

Examples: ROOT(100, 2) returns 10.0

ROOT(100, -2) returns 0.10

See also: POWER, SQRT

ROUND(x)

Returns: *x* rounded and returned as the nearest integer.

Examples: ROUND(1.9) returns 2

ROUND(2.0) returns 2

ROUND(1.5) returns 2

ROUND(-1.5) returns -2

See also: FRAC, CEIL, INT, FLOOR, TRUNC

RRAND(x, y)

Returns: A real random number from *x* to *y*.

Notes: Before RRAND or other random number functions can be used, the value of RANDOMSEED must be set to a positive constant.

See also: RAND, IRAND

RTOD(x)

Returns: Radians from degrees, $180x/\pi$.

Examples: RTOD(0.5) returns 28.647889756541

RTOD(PI) returns 180.0

See also: DTOR

SEC(x)

Returns: The secant of *x*.

SECH(x)

Returns: The hyperbolic secant of *x*.

SGN(x)

Returns: 1 if $x > 0$, 0 if $x = 0$, or -1, if $x < 0$.

Examples: SGN(3) returns 1

SGN(0) returns 0

SGN(-2) returns -1

See also: SIGN, DELTA, HEAVISIDE

SHIFTLEFT(x, y)

Returns: Shifts bits of *x* to the left by *y* binary positions.

Examples: SHIFTLEFT(1, 5) returns 32

SHIFTLEFT(2, 10) returns 2048

See also: SHIFTRIGHT

SHIFTRIGHT(x, y)

Returns:	Shifts bits of *x* to the right by *y* binary positions.
Examples:	SHIFTRIGHT(32, 5) returns 1
	SHIFTRIGHT(2048, 10) returns 2
See also:	SHIFTLEFT

SIGN(x, y)

Returns:	*x* with the same sign as *y*.
Examples:	SIGN(-234, 1) returns 234
	SIGN(234, 0) returns 234
	SIGN(-234, 0) returns 234
	SIGN(234, -1) returns -234

SIN(x)

Returns:	The sine of angle (in radians) *x*
Examples:	SIN(PI) returns 0.0
	SIN(PI/2) returns 1.0

SINH(x)

Returns:	The hyperbolic sine of *x*.
Examples:	SINH(1) returns 1.1752011936438
	SINH(0) returns 0.0
	SINH(-1) returns -1.175201193644

SPHERETORECTX(r, a1, a2)

Returns:	Rectangular *x* from 3 spherical coordinates, radius *r*, angles (in radians) *a1* and *a2*.
Examples:	SPHERETORECTX(1, PI, PI) returns 0
	SPHERETORECTX(1, PI/4, PI/4) returns 0.5
See also:	SPHERETORECTY, SPHERETORECTZ, RECTTOSPHERER, RECTTOSPHEREA1, RECTTOSPHEREA1, RECTTOPOLARA, RECTTOPOLARR, POLARTORECTX, POLARTORECTY

SPHERETORECTY(r, a1, a2)

Returns:	Rectangular *y* from 3 spherical coordinates, radius *r*, angles (in radians) *a1* and *a2*.
Examples:	SPHERETORECTY(1, PI, PI) returns 0
	SPHERETORECTY(1, PI/4, PI/4) returns 0.5
See also:	SPHERETORECTX, SPHERETORECTZ, RECTTOSPHERER, RECTTOSPHEREA1, RECTTOSPHEREA1, RECTTOPOLARA, RECTTOPOLARR, POLARTORECTX, POLARTORECTY

SPHERETORECTZ(r, a1, a2)

Returns:	Rectangular *z* from 3 spherical coordinates, radius *r*, angles (in radians) *a1* and *a2*.
Examples:	SPHERETORECTZ(1, PI, PI) returns -1
	SPHERETORECTZ(1, PI/4, PI/4) returns 0.7071067811865
See also:	SPHERETORECTX, SPHERETORECTY, RECTTOSPHERER, RECTTOSPHEREA1, RECTTOSPHEREA1, RECTTOPOLARA, RECTTOPOLARR, POLARTORECTX, POLARTORECTY

SQR(x)

Returns:	*x* squared, x^2
Examples:	SQR(PI) returns 9.8696044010894
	SQR(10) returns 100
	SQR(-5) returns -25
See also:	POWER

SQRT(x)

Returns:	The square root of *x*, $\sqrt{x}$
Examples:	SQRT(100) returns 10
	SQRT(25) returns 5
See also:	ROOT

STANDARDIZE(x, μ, σ)

Returns:	A value for *x* standardized by location parameter μ and scale parameter σ, as $(x - \mu)/\sigma$.
Examples:	STANDARDIZE(1, 2, 1) returns -1.0

STANDARDIZE(1, 1, 1) returns 0.0

STANDARDIZE(2, 1, 1) returns 1.0

STRING2INT(s)

Returns: An integer number from string *s*

Examples: STRING2INT("-124") returns -124

STRING2INT("0rMCMXIV") returns 1914

See also: STRING2REAL

STRING2REAL(s)

Returns: A real number from string *s*

Examples: STRING2REAL("-124.73") returns -124.73

STRING2REAL("0rMCMXIV") returns 1914.0

See also: STRING2INT

SUBSTRING(x, y, z)

Returns: A substring from string *x*, beginning in position *y* and length *z*.

Examples: SUBSTRING("Probability theory", 5, 7) returns "ability"

SUBSTRING("Small", 10, 5) returns the empty string ""

SUBSTRING("Small", 2, 20) returns "mall"

See also: LEFTSTRING, RIGHTSTRING

SUBTRACT(x, y)

Returns: $x - y$. This is the functional form of the algebraic $x - y$.

Examples: SUBTRACT(10, 2) returns 8

SUBTRACT(-10, 2) returns -12

See also: ADD

TAN(x)

Returns: The tangent of angle (in radians) *x*.

Examples: TAN(DTOR(45)) returns 1.0
TAN(0) returns 0

TANH(x)

Returns: The hyperbolic tangent of *x*.
Examples: TANH(0) returns 0
TANH(1) returns 0.7615941559558

TOLOWER(x)

Returns: A string in lower case.
Examples: TOLOWER("A STRING") returns "a string"
TOLOWER('A' + 'b') returns "ab"
See also: TOUPPER

TOUPPER(x)

Returns: A string in upper case.
Examples: TOUPPER("a string") returns "A STRING"
TOUPPER('A' + 'b') returns "AB"
See also: TOLOWER

TRIM(x)

Returns: A string with leading and trailing spaces removed.
Examples: TRIM(" a string ") returns "a string"
TRIM(" ") returns "" (empty string)
See also: TRIML, TRIMR

TRIML(x)

Returns: A string with leading spaces removed.
Examples: TRIML(" a string ") returns "a string "
TRIML(" ") returns "" (empty string)

See also: TRIM, TRIMR

TRIMR(x)

Returns: A string with trailing spaces removed.

Examples: TRIMR(" a string ") returns " a string"

TRIMR(" ") returns "" (empty string)

See also: TRIM, TRIML

TRUNC(x)

Returns: *x* truncated to an integer.

Examples: TRUNC(1.0) returns 1

TRUNC(2.8) returns 2

TRUNC(-2.5) returns -2

See also: ROUND, FRAC, CEIL, INT, FLOOR

WEEKDAY(x)

Returns: A numerical day of the week (Sun=1 Mon=2...) for Julian day *x*.

Examples: WEEKDAY(DMYTOJ(01, 01, 2000)) returns 7 (Saturday)

WEEKDAY(DMYTOJ(15, 01, 2001)), M.L. King Jr.'s birthday, returns 2 (Monday).

See also: See also: JULIAND, JULIANM, JULIANY, YEARDAY, DMYTOJ

XORF(x, y)

Returns: Logical or boolean XOR function. This is the functional from of *x* XOR *y*.

Examples: XORF(TRUE, TRUE) returns FALSE

XORF(FALSE, FALSE) returns FALSE

XORF(TRUE, FALSE) returns TRUE

XORF(2x010101, 2x000111) returns 18 (2x010010)

Notes: If both *x* and *y* are integer types, XORF(*x*, *y*) returns the bitwise (logical) XOR of the two numbers. If *x* and *y* are boolean types, XORF(*x*, *y*) returns the boolean XOR of the two numbers.

See also ORF, NOTF, ANDF

YEARDAY(x)

Returns:	Day of the year (1-jan = 1...) for Julian day *x*.
Examples:	YEARDAY(DMYTOJ(01, 01, 2000)) returns 1
	YEARDAY(DMYTOJ(30, 06, 2000)) returns 182
See also:	See also: JULIAND, JULIANM, JULIANY, WEEKDAY, DMYTOJ

Calculator mode

mle can be made to act like a calculator. In this mode, instead of a program filled with assignment statement, data statements, and model statements, a series of expressions are given to ***mle***. The expressions are evaluated and the result is printed. This can be done either interactively (using the `-i` command line option) or by reading in a program file.

This "calculator" mode is invoked by not including the prefix `mle` as the first thing read either interactively or in the program file. ***mle*** will then execute all subsequent commands as expressions to be interpreted. The exception to this is that `DATA` statements are also legal. Here is an example

```
c:\>mle -i
sin(pi * 3)                                  This is the user-defined expression
2.168404E-0019                               And this is what was returned

PDF normal(2, 3) 1, 2 end                    Compute the area under normal pdf from 2 to 3, μ=1, σ=2
0.1498822726114                              resulting area

INTEGRATE z (2, 3) PDF NORMAL(z) 1, 2 end end   Expressions can be nested. Integrate for 2 to 3 a normal pdf with
μ=1, σ=2
0.1498822847945                              This should be close to the previous result

gamma(3.8)                                   Evaluates the gamma function
4.6941742051124
summation i (1, 10) 1/i^2 end                Sum from 1 to 10, 1/i^2
1.5497677311665

end                                          Ends and returns to DOS
```

In version 2 of ***mle***, when using calculator mode interactively, there will always be a delay of one expression before the results is returned. This is because an expression can continue on indefinitely. For example, the expression "`SIN(2*pi)`" followed by a carriage return does not complete the expression because the next line may be "`+ 1/2`". A new expression is needed to denote the end of the old expression. Thus, typing "`1 pi 2`" followed by a carriage return will result in two complete expressions (returning `1` and `3.1415926535898`). The third expression is not yet complete.

Note that if you begin ***mle*** with the options `-i -v` and begin typing expressions, the verbose result will be to show the entire expression in functional form (i.e. as a series of functions). For example

```
c:\>mle -i -v
sin(pi^2/4 + 1)                              This is the user-defined expression
```

returns

```
SIN(ADD(DIVIDE(POWER( PI ,  2),  4),  1)) ->  -0.320074806512
```

SOME EXAMPLE PROGRAMS

Survival analysis—Exact measurements

This first example not only provides an illustration of a simple ***mle*** program, but also shows the notation that will be used throughout this chapter. The problem at hand is finding one or more parameters $\boldsymbol{\theta}$ of some distribution $f(t|\boldsymbol{\theta})$, given a series of observations, $\mathbf{t}=t_1, t_2, \ldots, t_N$. The values of $\mathbf{t}$ are known exactly. For an individual observation, t_i, the individual likelihood is $L_i = f(t_i|\boldsymbol{\theta})$, and the overall likelihood for the N observations is

$$L(a,b\,|\,\mathbf{t}) = \prod_{i=1}^{N} f(t_i\,|\,a,b)dt\,. \tag{2}$$

Data for this example (Table 14) are a series of 15 observations of times to breakdown for an insulating fluid at 32 kV. The times are arranged as one observation per line in a file named ex1.dat. The underlying distribution is believed to follow a negative exponential probability density function, with a single parameter lambda. The following ***mle*** program analyses these data

```
MLE
  TITLE = "32 kV Insulating Fluid Example from Nelson (1982:105)"
  DATAFILE("ex1.dat")    {Input data file name}
  OUTFILE("ex1.out")     {Name to which results are written}

  DATA
    failtime   FIELD 1
  END

  MODEL
    DATA
       PDF EXPONENTIAL(failtime)
         PARAM  lambda  LOW=0.00001  HIGH=1  START=0.05  END
       END  {of the PDF}
    END
  RUN
    FULL
  END  {of the MODEL}
END    {of the MLE program}
```

Here is the abridged output

Table 14 Times to breakdown for an insulating fluid at 32 kV, from Nelson W (1982:105).

0.27	0.4	0.69
0.79	2.75	3.91
9.88	13.95	15.93
27.8	53.24	82.85
89.29	100.58	215.1

```
New model:  32 kV Insulating Fluid Example

LogLike= -70.76273 Iterations= 2 Func evals= 26 Del(LL)= 0.0000000000
Converged normally

Results with estimated standard errors.  (7 evals)
Solution with 1 free parameter
         Name Form       Estimate         Std Error          t        against
       lambda LOGLIN   0.024294254090   0.004468859626    5.43634307759   0.0

Likelihood CI Results:  (21 evals)
Solution with 1 free parameter
         Name Form       Estimate          Lower CI          Upper CI
       lambda LOGLIN   0.024294254090   0.009423459169    0.049934238797
```

The first part of the output shows the loglikelihood, and information about iterations, function evaluations, and convergence. This is followed by two output reports, first with standard errors and the with an approximate likelihood confidence interval (region).

Survival analysis—Exact failure and right censored observations

The standard problem in survival analysis is to find parameters of a parametric model when some observations are right censored. Typically we have N exact observations, and N^+ right-censored observations, the likelihood is

$$L(\boldsymbol{\theta}\,|\,\mathbf{t}) = \prod_{i=1}^{N} f(t_i\,|\,\boldsymbol{\theta}) \prod_{i=1}^{N^+} S(t_i\,|\,\boldsymbol{\theta}), \tag{3}$$

where $S(t|\boldsymbol{\theta})$ is the survival distribution, which is the area under $f(t|\boldsymbol{\theta})$ to the right of t. The area under a right censored observation is specified in the ***mle*** `PDF` function by setting the second time variable to infinity (or something less than the first time variable). So, the function `PDF NORMAL(14,-1) 10, 6 END` would return the area from 14 to infinity of under a normal pdf with parameters $\mu = 10$, and $\sigma = 6$, or about 0.2525. This would correspond to the likelihood of an individual surviving to 14 units of times under the specified model.

For this example, we use the data in Table 14 and suppose that there were three additional observations that had not failed by time 220—the end of the experiment. The data will be coded so that the three right censored times are given as negative times, -220. The `DATA` statement now creates two variables, the first is the absolute value of time to failure, and the second is the unmodified time. Thus, failed observations have two identical failure times, for example [9.88, 9.88], which defines an exact failure; whereas, right-censored observations have a positive and a negative censored time [220, -220], yielding the area under the pdf from 220 to infinity.

```
MLE
  TITLE = "32 kV Insulating Fluid Example"
  DATAFILE("ex2.dat")    {Input data file name}
  OUTFILE("ex2.out")     {Name to which results are written}

  DATA
    topen   FIELD 1 = ABS(topen)
    tclose  FIELD 1
  END

  MODEL
    DATA
       PDF EXPONENTIAL(topen, tclose)
         PARAM  lambda  LOW=0.00001  HIGH=1  START=0.05  END
       END  {of the PDF}
    END
  RUN
    FULL
  END  {of the MODEL}

END    {of the MLE program}
```

The abridged output is

```
18 lines read from file ex2.dat
18 Observations kept and 0 observations dropped.

New model:  32 kV Insulating Fluid Example

LogLike= -81.66833 Iterations= 2 Func evals= 28 Del(LL)= 0.0000000000
Converged normally

Results with estimated standard errors.  (8 evals)
Solution with 1 free parameter
         Name Form       Estimate         Std Error          t        against
       lambda LOGLIN   0.011742333138   0.002142967492    5.47947329296   0.0

Likelihood CI Results:  (21 evals)
Solution with 1 free parameter
         Name Form       Estimate          Lower CI          Upper CI
       lambda LOGLIN   0.011742333138   0.004554712392    0.024135096958
```

Survival analysis—Interval censored observations

Interval censored observations, are those collected between two points of time. These observations frequently arise from prospective studies in which periodic observations are collected. The exact times to the event are not known. What is known is t_u, the last time before the event occurred, and t_e, the time of the first observation after the event occurred. The likelihood for interval censored events is the area under the pdf between t_u and t_e,

$$L(\boldsymbol{\theta} \mid \mathbf{t}_u, \mathbf{t}_e) = \prod_{i=1}^{N} \left[S(t_{u_i} \mid \boldsymbol{\theta}) - S(t_{e_i} \mid \boldsymbol{\theta}) \right] \tag{4}$$

In ***mle***, the area under the pdf is specified for most distributions as the first two times, with the second time greater than the first. For example, `PDF NORMAL(11, 15) 10, 6 END` return 0.231, which is the area between 11 and 15 under a normal distribution with μ=10, and σ=6. Here is an ***mle*** program that finds parameters of a lognormal distribution from interval censored data.

```
MLE
  TITLE = "Example"
  DATAFILE("ex3.dat")
  OUTFILE("ex3.out")

  DATA
    topen   FIELD 1
    tclose  FIELD 2
  END

  MODEL
    DATA
       PDF LOGNORMAL(topen, tclose)
         PARAM  a  LOW=0.00001  HIGH=9  START=1  END
         PARAM  b  LOW=0.00001  HIGH=2   START=0.4  END
       END  {of the PDF}
    END
  RUN
    FULL
  END  {of the MODEL}

END
```

Current status analyses

Current status analysis consists of observations that are collected cross-sectionally. The methods most commonly associated with current status analysis is probit and logit analysis. ***mle*** allows for current status analysis with any of the built-in distribution functions.

Under a cross-sectional study design, each observation consists of (1) time of a single observation since the study began (t), (2) an indicator variable to determine whether or not the individual experienced the event. The result of the indicator variable is that the individual is a responder (r) or non-responders (n). The likelihood from N observations made up of N_r responders and N_n non-responders is

$$L(\boldsymbol{\theta}\,|\,\mathbf{t}) = \prod_{i=1}^{n} S(t_i\,|\,\boldsymbol{\theta}) \prod_{i=1}^{r} F(t_i\,|\,\boldsymbol{\theta}) \tag{5}$$

This likelihood can be interpreted as follows. For the likelihood for the non-responders is the area under the pdf from the time of observation to infinity. Thus, a responder contributes a likelihood that is exactly like a right-censored observation. The likelihood for a responder is the area under the pdf from -∞ (or 0 for pdfs defined to have positive arguments) to the time of observation, which is the probability of the event occurring at some time unknown time before the time of observation. In mle, the area under the likelihood for a responder is specified as `PDF LOGNORMAL(-1, 5) 2, 0.5 END` return 0.217, which is the area between 0 (or anything less than 0) and 5 under a lognormal distribution with μ=2, and σ=0.5.

Consider a data set that contains a time of observation and an indicator variable that is 0 if the observation was a non-responder and 1 for a responder. One way of coding this model is to place an `IF...THEN...ELSE...END` statement to switch between responder and nonresponder likelihoods as appropriate for each observation:

```
MLE
  TITLE = "Example"
  DATAFILE("ex4.dat")
  OUTFILE("ex4.out")

  DATA
    t       FIELD 1  {time of observation}
    resp    FIELD 2  {1 if responder, 0 if nonresponder}
  END

  MODEL
    DATA
      IF resp = 1 THEN        {it is a responder}
        PDF LOGNORMAL(0, t)
          PARAM  a  LOW=0.00001  HIGH=9  START=1    END
          PARAM  b  LOW=0.00001  HIGH=2  START=0.4  END
        END  {of the PDF}
      ELSE  {non-responder}
        PDF LOGNORMAL(t, oo) a, b END
      END  {of if then else}
    END {data}
  RUN
    FULL
  END  {of the MODEL}

END
```

Alternatively, The following ***mle*** data statement will transform the observation time into a set of two times. For a responder, `topen` will be set to zero and `tclose` will take the value of the observed time. For a non-responder, `topen` will take the value of the observed time and `tclose` will be set to zero. Note that when the second time is set to zero, it will be less than `topen`, so ***mle*** returns the area from `topen` to infinity.

```
MLE
  TITLE = "Example"
  DATAFILE("ex4.dat")
  OUTFILE("ex4.out")

  DATA
    time     FIELD 1    {read in observation time}
    resp     FIELD 2    {1 if responder, 0 if nonresponder}
    topen     = IF resp == 1 THEN 0 ELSE time END
    tclose    = IF resp == 1 THEN time ELSE -1 END
  END

  MODEL
    DATA
       PDF LOGNORMAL(topen, tclose)
         PARAM  a  LOW=0.00001  HIGH=9  START=1    END
         PARAM  b  LOW=0.00001  HIGH=2  START=0.4  END
       END  {of the PDF}
    END
  RUN
    FULL
  END  {of the MODEL}

END
```

Survival analysis—With left-truncated observations

Left truncation arises in survival analysis when some early portion of an individual's period of risk is not observed. For example, in a prospective study of mortality, we might want to follow all living people in some area, instead of just following individuals from birth. This type of data collection can lead to unbiased results, provided observations are left-truncated at the age at which people are enrolled in the study. The idea is that, had the someone died prior to being enrolled in the study, that would not have been enrolled; therefore, their risk of mortality is know to be zero.

For this example, we will use the Siler competing hazards mortality model for a fictitious prospective study of mortality. We will two types of observations: those who died and those who are right censored. For each observation we know three times: the time an individual was enrolled for prospective observation (t_α), the last time an individual was observed as alive (t_u), and the first time the individual was known to be dead (t_e). The first time, t_α, defines the left truncation point, t_u and t_e define an interval within which death took place. For right censored observations, t_e is set to infinity (or a number greater than the human lifespan). The likelihood is

$$L(\boldsymbol{\theta}, \mathbf{t}_u, \mathbf{t}_e, \mathbf{t}_\alpha) = \prod_{i=1}^{N} \frac{S(t_{u_i} \mid \boldsymbol{\theta}) - S(t_{e_i} \mid \boldsymbol{\theta})}{S(t_{\alpha_i} \mid \boldsymbol{\theta})}. \tag{6}$$

From this likelihood it can be seen that an individual's probability of death is the area under pdf between t_u and t_e and divided by the area from t_α to infinity, which renormalizes the pdf for the period of actual observation. An individual likelihood is constructed in ***mle*** as `PDF SILER(14, 15, 6) 0.05, 0.3, 0.0, 0.001, 0.05 END`, which represents a person who died between ages 14 and 15, and were enrolled in the study at age 6.

```
MLE
  TITLE = "Example"
  DATAFILE("ex5.dat")
  OUTFILE("ex5.out")

  DATA
    talpha    FIELD 1  {Left truncation time}
    topen     FIELD 2  {time last known alive}
    tclose    FIELD 2  {time first known dead, or oo if censored}
  END

  MODEL
    DATA
       PDF SILER(topen, tclose, talpha)
         PARAM  a1  LOW=0.00001  HIGH=0.5 START=0.01   END
         PARAM  b1  LOW=0.01     HIGH=2   START=0.1    END
         PARAM  a2  LOW=0        HIGH=1   START=0.001  END
         PARAM  a3  LOW=0.0000   HIGH=1   START=0.001  END
         PARAM  b3  LOW=0.00001  HIGH=1   START=0.001  END
       END  {of the PDF}
    END
  RUN
    FULL
  END  {of the MODEL}

END
```

Survival analysis—right-truncated observations

Right truncation arises in survival analysis when the later risk is determined by the study design. For example, we might have data on child mortality for analysis. Each child was followed from birth to age five, and the only children available in the data set were those who died from birth to five. This type of data collection can lead to unbiased results, provided child's observations are right-truncated at age five.

For this example, we will use the Gompertz competing hazards mortality model for a fictitious prospective study of mortality. We will have observations selected for mortality by age five and no right-censoring. A single age at death is known. The likelihood for exact times to death with right truncation is

(7) $$L(\boldsymbol{\theta}, \mathbf{t}, \mathbf{t}_{\omega}) = \prod_{i=1}^{N} \frac{f(t_i \mid \boldsymbol{\theta})}{1 - S(t_{\omega_i} \mid \boldsymbol{\theta})}.$$

From this likelihood it can be seen that an individual's probability of death is the pdf at the age of death, divided by the area from 0 to t_{ω}, which renormalizes the pdf for the period of actual observation. An individual likelihood is constructed in ***mle*** as `PDF GOMPERTZ(2.1, 2.1, 6) 0.05, 0.3 END`, which is a death at the age of 2.1.

```
MLE
  TITLE = "Example"
  DATAFILE("ex6.dat")
  OUTFILE("ex6.out")

  DATA
    tdeath    FIELD 1  {Left truncation time}
  END

talpha = 5.0     {set a constant for right truncation}

  MODEL
    DATA
       PDF GOMPERTZ(tdeath, tdeath, talpha)
         PARAM  a1  LOW=0.00001  HIGH=0.5  START=0.01  END
         PARAM  b1  LOW=-2       HIGH=-0   START=0.1   END
       END  {of the PDF}
    END
  RUN
    FULL
  END  {of the MODEL}

END
```

Survival analysis—With left-and right-truncated observations

This example extends the previous one by including both left and right truncation, as well as interval censored observations. We will use a child mortality example again, but now each children is recruited at some age from 0 to 5 years. Their risk will be left-truncated at the age of entry. Again, only children who die before age 5 would be included in the analysis, so that all exposures are right-truncated. Finally, children are periodically visited, so all observations are interval censored. Again, we will use the Gompertz competing hazards mortality model for this fictitious prospective study of child mortality. The likelihood is

(8) $$L(\boldsymbol{\theta}, \mathbf{t}_u, \mathbf{t}_e, \mathbf{t}_{\alpha}, \mathbf{t}_{\omega}) = \prod_{i=1}^{N} \frac{S(t_{u_i} \mid \boldsymbol{\theta}) - S(t_{e_i} \mid \boldsymbol{\theta})}{S(t_{\alpha_i} \mid \boldsymbol{\theta}) - S(t_{\omega_i} \mid \boldsymbol{\theta})}.$$

From this likelihood it can be seen that an individual's probability of death is the area under pdf between t_u and t_e and divided by the area from t_{α} to t_{ω}, which renormalizes the pdf for the period of actual observation. An individual likelihood is constructed in ***mle*** as `PDF GOMPERTZ(topen, tclose, talpha, tomega) 0.05, 0.3 END`. For example `PDF GOMPERTZ(2.1, 2.4, 1.0, 5.0) 0.05, 0.3 END` returns the probability that a child, enrolled in the study at age one and selected for having died by age five, died between the ages of 2.1 and 2.4.

```
MLE
  TITLE = "Example"
  DATAFILE("ex7.dat")
  OUTFILE("ex7.out")

  DATA
    talpha    FIELD 1  {Left truncation time}
    topen     FIELD 2  {time last known alive}
    tclose    FIELD 2  {time first known dead, or oo if censored}
  END

tomega = 5.0

  MODEL
    DATA
       PDF GOMPERTZ(topen, tclose, talpha, tomega)
         PARAM  a1  LOW=0.00001  HIGH=0.5 START=0.01  END
         PARAM  b1  LOW=0.01     HIGH=2   START=0.1   END
       END  {of the PDF}
    END
  RUN
    FULL
  END  {of the MODEL}

END
```

Survival analysis—Accelerated failure time

Frequently, one is interested in modeling the effects of covariates on the time to failure. A common model of this type is call the accelerated failure time model (AFT), in which covariates shift the time to failure to the right or the left. ***mle*** has a general mechanism for modeling the effects of covariates on any parameter that is defined, so that accelerated failure time models can be easily constructed.

In this example, the mean of a normal distribution has two covariates that shift the failure time.

```
MLE
  TITLE = "Example"
  DATAFILE("ex8.dat")
  OUTFILE("ex8.out")

  DATA
    topen     FIELD 1  {Last observation time prior to the event}
    tclose    FIELD 2  {First observation time after the event}
    weight    FIELD 3  {the first covariate}
    age       FIELD 4  {the second covariate}
  END

  MODEL
    DATA
       PDF NORMAL(topen, tclose)
         PARAM  mu  LOW=0.00001  HIGH=100 START=25  FORM=LOGLIN
            COVAR weight PARAM b_weight LOW=-20 HIGH=20  START=0  END
            COVAR age    PARAM b_age    LOW=-20 HIGH=20  START=0  END
         END     {param mu}
         PARAM  s  LOW=0.01  HIGH=50  START=3  END
       END  {of the PDF}
    END
  RUN
    FULL
  END  {of the MODEL}

END
```

From this specification of covariates, the μ intrinsic parameter of the normal distribution will be computed for the *i*th observation as $\mu_i = \texttt{mu} \times \exp(\texttt{weight}_i \times \texttt{b\_weight} + \texttt{age}_i \times \texttt{b\_age})$.

Survival analysis—Hazards model

An alternative to the accelerated failure time model is the hazards model. Under the hazards model, the effects of covariates is to raise or lower the hazard by some amount[9]. In general, if $h(t)$ is the hazard function, covariates for the ith individual, $x_i\beta$, are modeled on the hazard as $h_i(t) = h(t)\exp(x_i\beta)$.

Most of the probability density functions in ***mle*** provide a mechanism for modeling the effects of covariates on the hazard. You can find out for any particular pdf by typing, for example, `mle -h lognormal`. A message will tell you whether or not covariates can be modeled on the hazard.

In this example, the same normal distribution used in the previous example has had the two covariates moved from affecting μ to affecting the hazard.

```
MLE
  TITLE = "Example"
  DATAFILE("ex8.dat")
  OUTFILE("ex8.out")

  DATA
    topen     FIELD 1  {Last observation time prior to the event}
    tclose    FIELD 2  {First observation time after the event}
    weight    FIELD 3  {the first covariate}
    age       FIELD 4  {the second covariate}
  END

  MODEL
    DATA
       PDF NORMAL(topen, tclose)
         PARAM  mu  LOW=0.00001  HIGH=100  START=25  END
         PARAM  s   LOW=0.01     HIGH=50   START=3   END
         HAZARD COVAR weight PARAM b_weight LOW=-20 HIGH=20  START=0  END
                COVAR age    PARAM b_age    LOW=-20 HIGH=20  START=0  END
         END     {hazard}
       END  {of the PDF}
    END
  RUN
    FULL
  END  {of the MODEL}

END
```

Survival analysis—Immune subgroup

When observing times to events, there may be an unidentifiable subgroup for whom risk of experiencing the event is zero. These make up a so-called *immune fraction*, a *sterile subgroup*, or a *contaminating fraction*. It is possible to model some fraction of individuals who are not at risk, so to statistically identify the subgroup.

If complete records are available for all individuals, one could simply remove the sterile individuals from the analysis of the non-sterile fraction. When complete records are not available (i.e. we cannot tell a sterile individual from a right-censored individual) maximum likelihoods methods are easily adapted to include estimation of an unknown fraction of individuals who are not susceptible to failure.

The effect of the sterile subgroup on the survival distribution can be seen in Figure 5. Call s the non-susceptible fraction. Then the proportion of individuals who are susceptible at the start of risk is $p(0)=1-s$. Inspection of Figure 5

[9] Except for the exponential and the weibull distributions, accelerated failure time models are not proportional hazards models.

suggests that the fraction of surviving individuals at time *t* must be made up of two fractions. One is $S_f(t)$ weighted by the fraction not sterile, $(1 - s)$. The second fraction is constant at *s*:

$$S(t) = (1-s)S_f(t) + s\,.$$

The overall hazard at time t is simply the hazard of the non-susceptible subgroup weighted by the proportion of that group at time *t*. The proportion of susceptible individuals at time *t* will decrease as fecund individuals fail, and must depend on survivorship of the non-sterile group to time *t* and the initial fraction of sterile individuals, *s*. This fraction at time *t* is

$$p(t) = \frac{(1-s)S_f(t)}{s + (1-s)S_f(t)}\,.$$

The hazard at time *t* is

$$h(t) = p(t)h_f(t) = \frac{(1-s)S_f(t)}{s + (1-s)S_f(t)}\frac{f_f(t)}{S_f(t)} = \frac{(1-s)f_f(t)}{s + (1-s)S_f(t)}$$

and the probability density function is found as

$$f(t) = h(t)S(t) = (1-s)S_f(t)h_f(t) = (1-s)f_f(t)\,.$$

These forms for the PDF, SDF, and hazard function provide for reasonably straight-forward maximum likelihood estimation of the parameters of the distribution for the susceptible observations as well as *s*. The general form of the likelihood when sterility is included, becomes

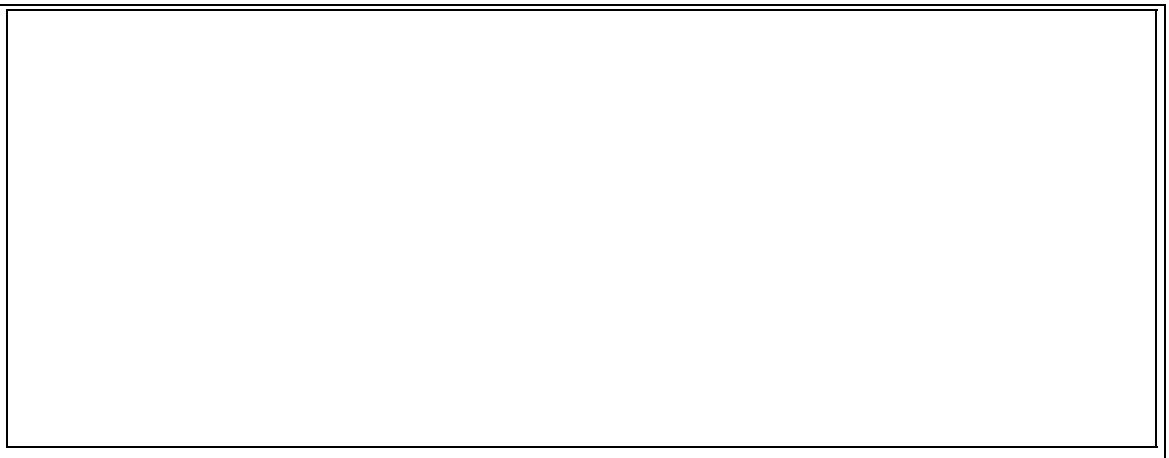

Figure 5. The effect of contamination by a sterile subgroup on the survivorship distribution. The subgroup makes up fraction *s* of the initial population at risk. The left panel shows survivorship for the uncontaminated group and the right panel shows the same distribution contaminated by the sterile subgroup.

$$(9)\quad L(\boldsymbol{\theta}, s \mid t_u, t_e) = \prod_{i=1}^{N}\left[(1-s)\left[f(t_{e_i} \mid \boldsymbol{\theta})\right]^{\delta\{t_{u_i}, t_{e_i}\}}\left[S(t_{u_i} \mid \boldsymbol{\theta}) - S(t_{e_i} \mid \boldsymbol{\theta})\right]^{1-\delta\{t_{u_i}, t_{e_i}\}} + s\delta\{t_{e_i}, t_{\omega_i}\}\right],$$

where the $\delta\{x,y\}$ is the Kronecker's delta function, which equals one when $x=y$, and zero when $x \neq y$.

The following example estimates one such model. The likelihood begins with the `mix()` function, which produces an average of the second and third arguments, weighted by first argument (which is a probability). The first PDF is `PDF STERILE() END`, which returns one if `tclose` is infinity or less than `topen`. Covariates are modeled on both the non-susceptible fraction as well as the hazard of the susceptible fraction.

```
MLE
  TITLE = "Example"
  DATAFILE("ex.dat")
  OUTFILE("ex.out")

  DATA
    topen     FIELD 1  {Last observation time prior to the event}
    tclose    FIELD 2  {First observation time after the event}
    weight    FIELD 3  {the first covariate}
    age       FIELD 4  {the second covariate}
  END

  MODEL
    DATA
       MIX( PARAM s LOW=-100  HIGH=100 START=0  FORM=LOGLIN    {define the immune fraction}
              COVAR  weight  PARAM b_s_weight  LOW=-20  HIGH=20  START=0  END
              COVAR  sex     PARAM b_s_sex     LOW=-20  HIGH=20  START=0  END
            END  {param s}

            PDF STERILE(topen, tclose) END,     {returns 1 for right censored observations}

            PDF LNNORMAL(topen, tclose)
              PARAM  a  LOW=0.00001  HIGH=100 START=25  END
              PARAM  b  LOW=0.01     HIGH=50   START=3  END
              HAZARD COVAR weight PARAM b_weight LOW=-20 HIGH=20  START=0  END
                     COVAR sex    PARAM b_sex    LOW=-20 HIGH=20  START=0  END
              END  {hazard}
            END  {of the PDF}
       )  {mix function}
    END
  RUN
    FULL
  END  {of the MODEL}

END
```

Linear regression in the likelihood framework

This example shows how linear regression is treated within the framework of likelihood models. The linear regression model with n covariates specifies that the value of the ith observation is a combination of a y intercept term (α) an additive covariate-parameter term ($x_{i1}\beta_1 + x_{i2}\beta_2 + ... + x_{in}\beta_n$) plus an error ($e_i$). Furthermore, distribution among all error terms (ε) is normally distributed with a mean of zero and a standard deviation of σ. The formal specification is:

$$y_i = \alpha + x_{i1}\beta_1 + x_{i2}\beta_2 + ... + x_{in}\beta_n + e_i$$

$$\varepsilon \sim N(0, \sigma)$$

Under the likelihood model, the equivalent specification can be given in a very different format.

$$Y \sim f(\mu_i, \sigma)$$
$$\mu_i = \alpha + x_{i1}\beta_1 + x_{i2}\beta_2 + \ldots + x_{in}\beta_n$$

The difference in the two specifications exemplifies the two different philosophies in the methods. Under regression, difference between each observation and the line specified by the parameters is treated as "error". Under the likelihood model, the observations are normally distributed, with a mean that is determined by a series of covariates.

The data for this example are fictitious. The third column contains the values of y_i, column 1 is x_{i1} and x_{i2}.

```
0.4 53 64
0.4 23 60
3.1 19 71
0.6 34 61
4.7 24 54
1.7 65 77
9.4 44 81
10.1 31 93
11.6 29 93
12.6 58 51
10.9 37 76
23.1 46 96
23.1 50 77
21.6 44 93
23.1 56 95
1.9 36 54
29.9 51 99
```

The following shows the output from a regression analysis

```
     VARIABLE               MEAN      STD. DEVIATION COEF. VARIAT.
Indept Variable: Y        76.17647059   16.63293154    0.21834736
Depent Variable:  1       11.07058824    9.74453467    0.88021833
Depent Variable:  2       41.17647059   13.43612339    0.32630585

 VAR.    COEFFICIENT   STD ERROR    T STATISTIC
Alpha   66.46540496
B(1)     1.29019050    0.34276468    3.76407073
B(2)    -0.11103677    0.24858973   -0.44666675

              SUM OF              MEAN         F
SOURCE       SQUARES      DF      SQUARE      RATIO
REGRESS.    2325.1795      2   1162.5897      7.7458
RESIDUAL    2101.2911     14    150.0922
TOTAL       4426.4706     16    276.6544

R SQUARE =     0.5253
STANDARD ERROR OF ESTIMATE =    12.251213
```

The following shows the ***mle*** code for the equivalent likelihood model. Notice that this program is similar to the accelerated failure time model, except that the form for modeling covariates on the mean is additive (`FORM = ADD`).

```
MLE
  TITLE = "Test regression"
  DATAFILE("eg.dat")
  OUTFILE("eg.out")

  DATA
    y       FIELD 3
    x1      FIELD 1
    x2      FIELD 2
  END

  MODEL
    DATA
      PDF NORMAL(y)
        PARAM  mu   LOW = 7   HIGH = 500   START = 50 FORM = ADD
          COVAR  x1  PARAM  b1  LOW=-10  HIGH=10  START=0  END
          COVAR  x2  PARAM  b2  LOW=-10  HIGH=10  START=0  END
        END   {param}
        PARAM  sig  LOW=0.1   HIGH=200   START=10  END
      END   {pdf}
    END   {data}
  RUN
    FULL
  END

END
```

The following output fragment shows the result from this model.

```
LogLike= -65.06725 Iterations= 334 Func evals= 25383 Del(LL)=  9.745E-0011
Converged normally

Results with estimated standard errors.  (27 evals)
Solution with 4 free parameters
         Name Form       Estimate         Std Error          t        against
           mu ADD      66.46589883575   9.596050356992    6.92638078825   0.0
           b1          1.290194199465   0.453901547297    2.84245384742   0.0
           b2          -0.11104975496   0.202022074279    -0.5496911927   0.0
          sig          11.11779472801   2.630810510011    4.22599601366   0.0
```

The results are nearly identical to the regression results presented earlier. All parameters of the model are given with a standard error.

SOME DETAILS

This chapter is a series of notes and technical details for *mle*.

Maximizers

mle has four methods for maximizing the likelihood function. Each of the methods has strengths and weaknesses for different types of functions. Understanding some of the details of each method is useful for deciding which to use for any given application. The following sections describe each of the maximizers and points out strengths and weaknesses of each. The behavior of some methods can be modified considerably by the user.

The maximization method is selected by setting the variable `METHOD`.

The overall goal of function maximization is to find the set of parameters that maximize a function. A simple analogy is to imagine that you are looking at a topographic map that codes altitude by color. You want to find the longitude and latitude coordinates (the "parameters") that will put you at the highest point on the map. By looking over the map, you may be able to quickly ascertain a mountain peak or some other maximum. In order to do this, however, you effectively scanned hundreds of thousands of points on the map until finding those places where the colors suggest the highest altitude. With a little more work, the highest peak is easily resolved. Visual evaluation of maximum elevation is easy and takes almost no time because the map shows the elevations evaluated at hundreds of thousands of points on the map, and our eyes can quickly scan those points. That is, each "function" evaluation was inexpensive—we merely had to look at a point to know its value. Now imagine that the map surface is covered by a piece of paper. You can only expose a tiny hole in the map in order to read the color at that point (that is, to evaluate the function at that point). Furthermore, each hole takes a long time to cut, perhaps minutes or hours. Then the question becomes this: how do we find the maximum elevation of the map in the shortest possible time? The map analogy will be used to understand how different computer algorithms finds the maximum of a likelihood surface.

Many different function maximization methods have been developed at least since Isaac Newton developed methods out of the calculus. Nevertheless, no single method has emerged as superior for all types of problems. In general, function maximization is easiest to do when information is available for the derivative of the function. A traditional way of finding maximum likelihood parameters for simple functions is to symbolically find the derivatives of the function with respect to each free parameter. Each partial derivative is set to zero. This set of equations is collectively called the likelihood equations. Since the derivatives are defined as the slope of the function, it follows that any place where all the partial derivatives go to zero must be a minimum or a maximum of the function. If practical, the likelihood equations are "solved"; that is, the sets of parameter values are analytically found that simultaneously yields zero for each of the partial derivatives. The maximum likelihood estimates for a parameter is found from a particular series of observations by simply applying that equation on the set of observations. Unfortunately, this method is difficult and non-general and, therefore, not practical for general-purpose maximization as found in *mle*. Advances in computer-assisted symbolic mathematics (packages like Maple and Mathematica)

may eventually prove this method feasible for many users, but the need for specialized mathematical knowledge and skills still limits this method. A general method must work for most types of likelihood functions, whether or not analytical derivatives are easy (or even possible) to find.

Another class of fast maximizers estimates derivatives numerically. These methods are not robust for complex surfaces with many local maxima. From some starting point, they tend to rush up to the top of the nearest local maximum. A given function may have one or many points where the derivatives goes to zero, so this method may not find the global maximum. Numerical derivatives have limitations resulting, in part, from the inaccuracy of real number representation in computers, so that a number of derivative-free methods have been developed. One clever method solves a two dimensional maximization problem by trying to enclose the maximum within a triangle. The triangle grows and shrinks based only on information from the three points of the triangle at a given step. A rather unsophisticated method alternates between maximizing the function first by longitude, using as many evaluations as needed to find the maximum longitude for a given latitude, and then does the same for latitude. By repeating this many times, a maximum (usually the global maximum) is found. Needless to say, this method can be very slow. Finally, a newer method has been developed that mimics natures own maximization method. The method can be slow, but seems to be as robust at finding the global maximum as any iterative method.

Conjugate gradient method

The conjugate gradient method searches through parameter space for combinations of parameters where the slope of the likelihood function goes to zero. Now, the computer numerically computes a slope (or gradient) using the equation $m_i = [f(x_i + \Delta x_i) - f(x_i]/\Delta x_i$, for parameters **x** and small values Δ**x**. This procedure uses the slopes (m_i) to figure out the next set of **x** under the idea that the slope will decrease as the maximum is approached (unless the surface is flat).

The conjugate-gradient method used in ***mle*** was developed by Powell (1964), Brent (1973), and further developed by Press et al. (1989). For problems of more then two free parameters, the conjugate gradient method is usually much faster than the direct method. Caution must be exercised when using this method. At times a local maximum is latched onto by the solver and the rest of the parameter space is excluded. Furthermore, some conditions can cause the maximizer to leap to another part of the surface, where a local minimum might be reached. For example, when maximizing a likelihood function that includes numerical integration, the tolerance in the integrator must be several orders of magnitude smaller than that of the solver, or else the error in integration can lead the solver astray.

Two forms of the conjugate gradient method are available, `METHOD=CGRADIENT1` and `METHOD=CGRADIENT2`.

Simplex

The simplex method is a derivative-free maximization method described by Nelder and Mead (1965) and popularized by Press et al. (1989). The method is set with `METHOD=SIMPLEX`.

Direct method

A simple method for finding a maximum is to consider only one dimension at a time. So, for our map, we would find the highest latitude for a given longitude by examining points along a line of longitude. We could use the method of bisection or even better ways to find the maximum along that line of longitude in the fewest number of evaluations (i.e. fewest holes). Once we have settled on a latitude, we can find the longitude of highest elevation along that latitude. We next go back and find a new latitude for the new longitude, etc. This is known as the direct method (Nelson 1983), and works well for some functions over a small number of dimensions. In fact, the method is usually more robust at finding a global maximum than the simplex or congugate gradient methods. Furthermore, it is easy to constrain the algorithm so that new parameter values never overstep the user-defined (or mathematically defined) limits—that is, it respects the boundaries of our map. Unfortunately, the number of function evaluations goes up as an exponent of the number of dimensions in the problem. When the number of parameters gets large, the solution is very slow in coming. Furthermore, some functions that have the maximum along a long narrow ridge at a 45° angle to the lines of longitude and latitude require a large number of tiny movements before reaching the maximum.

The direct method and is set by `METHOD=DIRECT`. It uses the `HIGH =` value and `LOW =` values to constrain all parameters (as discussed below). The `START =` values define the initial starting parameters.

The direct method uses Brent's (1973; see also Press et al. 1989) parabolic interpolation to find the maximum along a single direction (i.e. for a single parameter holding all other parameters constant). The maximizer uses the `HIGH =` value and `LOW =` value to define the extreme bounds of the problem. The `START =` value is the first "guess" at the maximum. A parabola is then fit through the set of three points, and the maximum is analytically computed. This procedure is repeated with the three points enclosing the maximum until the maximum in that dimension is found to some prespecified tolerance. There are three ways you can modify the Brent maximizer. First, the maximum number of iterations in a single dimension can be set with `BRENT_ITS` = value, which is sufficient for almost every function. The next modification is to change the value of `BRENT_MAGIC` to some other number. This number defines the interpolation point between two points of a parabola—the so-called golden mean of ancient Greece. With such a heritage, there is little reason to change it. Finally, the value `BRENT_ZERO` is an arbitrary tiny number used in place of zero for the difference of two equal function evaluations.

Simulated annealing method

The simulated annealing method is an exciting and relatively new idea in maximization. It was first proposed by Kirkpatrick et al. (1983) for combinatorial problems. The algorithm was further developed for functions of continuous variables by Corana et al. (1987) and refined by Goffe et al. (1994); both papers lucidity describe how the method works.

As a metal is heated to its melting point, it loses its crystalline organization. Then as it again cools, the crystalline pattern reemerges. When cooled slowly, a process called *annealing*, small crystals of metal rearrange themselves and join other crystals with maximum orderliness (or minimum energy). This occurs as random movements of atoms and groups of atoms eventually fall into an alignments that minimize gaps. Once these structured alignments arise, they form a larger crystal and are subsequently less likely to fall out of alignment. As the temperature drops and the atoms move around less, large overall changes in structure become less probable. When absolute zero is reached, the structure becomes fixed (at room

temperature, solid metals continue to anneal very slowly). Rapid cooling of the metal, called *quenching* in metallurgy because the metal is thrust into cool water or pickle, does not provide sufficient time for crystals to move about and organize. Thus, numerous vacancies and dislocations exist among many small crystals, and orderliness is minimal. Maximizing the crystalline order (or minimizing vacancies and dislocations) is done by cooling the metal very slowly and providing ample opportunity for the random crystal movements to fortuitously align themselves into more ordered structures.

The simulated annealing method attempts to mimic the physical process of annealing. An initial "temperature" is set, and a cooling rate is specified. New parameters are randomly chosen over a large range of the parameter space. As the temperature cools, smaller and smaller ranges of the parameter space are explored. Additionally, the maximizer will not always travel up hill. At any given temperature, a certain fraction of downhill moves will be taken so that local maxima will not trap the maximizer.

The advantage of simulated annealing over other methods is that it is very good at finding the global maximum, even in the presence of highly multimodal likelihood surfaces. The user can fine tune the behavior of the algorithm so that functions with complex topography can be searched more thoroughly for the maximum. Another advantage of simulated annealing is that it does not require computation of derivatives. In fact, simulated annealing can find the maximum of discontinuous functions and those otherwise without first derivatives. Finally, the simulated annealing algorithm is extremely simple and intuitive. The disadvantages of simulated annealing are that it usually takes from one to several orders of magnitude more function evaluations than do other methods and the user must have an understanding of the algorithm to set up initial parameters that lend themselves to efficient estimation. Sometimes it is worth experimenting to find the best combinations of input parameters to the simulated annealing algorithm so as to minimize the total number of function evaluations.

Simulated annealing begins at some user-defined temperature (T) and a user-defined rate of cooling (r). At the end of one cycle of annealing, the temperature is reduced as $T = T \times r$, and a new cycle of annealing is performed. Typically the temperature will be 1 for simple function to 100,000 for difficult functions, and it is cooled every cycle by $r = 0.85$. When the algorithm begins, the starting point is evaluated and becomes the best value, so far. Each iteration will then search the likelihood surface in a partially random way and always keep track of the best point so far. A single cycle of annealing (i.e. one iteration) consists of the following. First, a cycle of random movements is started. N_{rand} random steps are taken over one direction at a time. The maximum width of the random step for parameter i is controlled by the step length variable v_i. For our map example, this would correspond to evaluating N_{rand} randomly picked points along a line of longitude or latitude. Initially we would use the entire height and with of the map for the maximum step length. As each point is evaluated, we keep track of the overall best maximum. Any time we find a point higher than our current maximum, we move to that point and consider it our new starting point. But, if a lower point is found we *might* accept that point according to the Metropolis criterion (Metropolis et al. 1953) by which the point is accepted with probability $\exp(-\Delta l/T)$, where Δl is the difference between the current starting point and the downhill point we have just evaluated. In other words, we draw a uniform random number on [0, 1), and accept the move if that number is less than a negative exponential survival function of Δl, with parameter $1/T$. This criterion means that at high temperatures we will frequently accept downhill moves with large changes in the loglikelihood, but as temperature drops, downhill moves will only occur at small changes in the loglikelihood. After completing the N_{rand} movements and evaluations, we now adjust the maximum steplength vector $\boldsymbol{v}$. The reduction or increase in steplength is done according to the proportion of accepted and rejected movements by an algorithm described in detail

below. In short, the maximum step length is reduced or increased so that we can expect to accept about one half of all moves in the next cycle of random steps. Following this adjustment, a new cycle of random steps is initiated until a total of N_{adj} of these adjustments have been completed. Thus, after $N_{rand} \times N_{adj}$ function evaluations, a single iteration completes, and a new iteration is begun until convergence, the maximum number of iterations is reached, or the maximum number of function evaluations is reached.

The simulated annealing method is set by `METHOD=ANNEALING`. The method does use the `HIGH =` value and `LOW =` values to constrain all parameters (as discussed below). The `START =` values define the initial starting parameters. A number of other variables should be set with this method. Since the simulated annealing method uses random numbers, the user must set a random seed, by calling the procedure `SEED()` with a positive integer. The starting temperature is set with `SA_TEMPERATURE`. The default value is 1000.0, which is too high for all but extremely wild functions. It is difficult to know what a good starting temperature is for a function, but values under 100 empirically seem to work for all but the most topographically complicated likelihood functions. When a likelihood is to be solved multiple times on similar data sets, like when running on bootstrapped data sets, it is worth exploring a couple of different temperatures and monitoring the progress of the annealing by using the verbose (`-v`) option. In fact, watching the entire annealing process is useful for developing and understanding of the algorithm. The variable `SA_COOLING` controls the cooling rate, and is 0.85 by default. Too high a value will slow down cooling and may lead to unnecessary evaluations, whereas too low a value may resulting in (simulated) quenching. The number of steps of random parameter perturbation is set using `SA_STEPS`. The number of step length adjustments taken every iteration is controlled by `SA_ADJ_CYCLES`. Finally, the size of each step adjustment can be controlled by `SA_STEPLENGTH_ADJ`, but the default value of 2.0 usually works well.

The simulated annealing algorithm uses a different criterion for convergence than do the other solvers. An array of the best likelihoods of size `SA_EPS_NUMBER` (default is 4) is created and updated every iteration. Convergence is considered achieved when the likelihood for the current iteration differs from all `SA_EPS_NUMBER` likelihoods by the value of `EPSILON`.

Several other variables can be used for fine tuning of the simulated annealing algorithm, but there is rarely a need to mess with them. `SA_STEPLENGTH` is the initial step length for all parameters. Empirically, the starting step length value has little effect on the outcome of the maximizer. `SA_ALT_ADJUSTMENT` uses an alternative formula for adjusting the step length. `SA_ADJ_LOWERBOUND` defines a "null" area for which step length is not adjusted. If the proportion of accepted moves is greater than `SA_ADJ_LOWERBOUND` and is less than 1 – `SA_ADJ_LOWERBOUND`, the current steplength will continue to be used. See Corana et al. (1987) for more details.

Stopping criteria

There are three ways to terminate finding the solution of a model. The first way is to minimize the change in the log-likelihood to below some specified minimum value. You can specify this by setting, for example, `EPSILON=1E-8`. When the absolute difference between the log-likelihoods of the previous iteration and the current iteration falls below this value, the problem will be considered to have converged normally.

The second way of controlling the stopping criteria is by specifying the maximum number of iterations permissible. For example, setting `MAXITER=1000`, would stop searching for the maximum after 1,000 iterations, regardless of the change in the likelihood. Note that a single iteration is that over all dimensions.

The third stopping criterion is by specifying the maximum number of function evaluations permissible. You can specify, for example, MAXEVALS=10000, which would stop searching for the maximum likelihood after 10,000 evaluations of the likelihood.

Looping through methods

mle provides a mechanism to specify that different methods be used to solve the same likelihood. For example, you can set

```
METHOD1=DIRECT
MAXITER1=10
METHOD2=CGRADIENT1
MAXITER2=500
```

to begin the problem with the direct method and then switch to a conjugate gradient solver for the next 500 iterations. The variables METHOD, MAXEVALS, MAXITER, and EPSILON can have a digit appended in this way. When the variable METHOD_LOOP is set to true, *mle* will loop back to the first method and continue the solver sequence again until one of the methods converges normally.

Output options

Options are provided for controlling the output format of the DATA and the MODEL statement. Many of the variables that control output options are boolean variables that are set to TRUE or FALSE.

DATA reports

For DATA statements, the values of all variables can be printed, summary statistics can be printed, and other information about reading and dropping observations can be printed. The PRINT_BASIC variable, when TRUE directs that the title, parameter file name, input file name, and the count of variables to be read from the input file are printed. The PRINT_FIELDS variable, when TRUE, prints out the name of each variable and the field it is read in from the input file.

The variable PRINT_DATA_STATS, when set to TRUE, prints summary statistics for each variable, including the mean, variance, standard deviation, minimum and maximum. When PRINT_OBS=TRUE, each observation is printed in the output file. PRINT_COUNTS, when TRUE, prints out how many lines were read from the input file, how many observations were kept, and how many observations were dropped.

MODEL reports

The output report from *mle* following the MODEL statement consists of parameter reports, the variance-covariance matrix, a list of the individual likelihoods for each observation, and plots of distributions.

Standard error report

A report with estimated standard errors is printed when `PRINT_SE = TRUE`. When the variable `PRINT_SHORT = TRUE`, the report format is modified so that all parameters estimates are printed on one line. Whenever standard error are reported, a variance-covariance matrix will be estimated. The next section discusses the details of computing and printing that matrix.

Variance-covariance matrix

An estimate of a variance-covariance matrix can be computed for the parameters by setting `PRINT_VCV = TRUE`. The number of elements of the matrix printed on a single line is normally 5, but can be changed by changing the value of `VCV_WIDTH`.

The asymptotic variance-covariances of maximum likelihood estimates is found by inverting the local Fisher's information matrix *I* for the *n* parameters:

$$I = \begin{bmatrix} E\left(\frac{-\partial^2 l}{\partial \theta_1^2}\right) & \cdots & E\left(\frac{-\partial^2 l}{\partial \theta_1 \theta_n}\right) \\ \vdots & \ddots & \vdots \\ E\left(\frac{-\partial^2 l}{\partial \theta_n \theta_1}\right) & \cdots & E\left(\frac{-\partial^2 l}{\partial \theta_n^2}\right) \end{bmatrix}$$

The expectations should be taken at the true parameter values. When parameter estimates cannot be evaluated analytically, numerical estimates of the information matrix, $\hat{\mathrm{I}}$ can be formed by plugging in parameter estimates, $\hat{\mathrm{q}}$. An estimated variance-covariance matrix is taken as $\hat{\mathbf{V}} = \hat{\mathbf{I}}^{-1}$.

mle uses two different estimates for the variance-covariance matrix. Either one, or both, methods may be used by setting `INFO_METHOD1` or `INFO_METHOD2` to `TRUE` or `FALSE`. The default method (`INFO_METHOD1`) computes the variance and covariance matrix by inverting Nelson's (1983) approximation to the Fisher's information matrix. The *x*th, *y*th element of that matrix is computed as $\hat{\mathrm{E}}_{xy} = \prod_i \left(\partial L_i / \partial x\right)\left(\partial L_i / \partial y\right)$, using the standard perturbation method for approximating the partial derivative. Appropriate sizes for Δx and Δy are iteratively computed for each parameter. ***mle*** initially uses a Δx (and Δy) of `DX_START` and then iteratively finds a Δx that changes the loglikelihood by at least `DX_TOOSMALL` but no more than `DX_TOOBIG`. Up to `DX_MAXITS` such iterations are permitted. The default values are almost always suitable. The one serious limitation of this method is that it does not work for hierarchical likelihoods.

The second estimate of the variance-covariance matrix is computed by estimating the second partial derivative by numeric perturbation. This method does not truly compute an expectation, and is inaccurate in some cases (you can compare the two methods by setting `INFO_METHOD2=TRUE`). Nevertheless, when hierarchical likelihoods are being computed, this method will produce better estimates.

Confidence interval report

An approximate confidence region for each parameter can be estimated by ***mle***. When the variable `PRINT_SHORT = TRUE`, the report format is modified so that all parameters estimates are printed on one line.

The confidence interval is defined as the extent of upper and lower perturbations away from the estimates that change the loglikelihood by a specified amount. For example, approximate 95% confidence intervals are formed when the change in the loglikelihood in each direction is 5.0239. This value corresponds to an expected probability of 0.025 on each tail of the chi-squared distribution with one degree of freedom. Over both directions, the total interval can be considered a 95% confidence interval for the parameter.

The interpretation of the one-dimensional confidence region must be done with caution. Figure 6 shows what happens when parameters are correlated (which is quite common). Panel a. shows the contour of the loglikelihood surface when parameter 1 is changed over the p_1 axis, and parameter 2 is changed over the p_2 axis. The bold ellipses represents the desired confidence level (say, 95%). The dotted lines show the confidence limits when p_1 is perturbed along the axis to each side of the estimate; this occurs where the bold ellipse intersects the p_1 axis. Panel *b*. shows what happens when parameters are correlated. Now, the dotted lines still show the 95% confidence limits when p_1 is perturbed from the estimate and p_2 is held constant at its maximum. The dashed lines show the true confidence region defined as the greatest extend of the 95% confidence ellipse over the space of p_1 and p_2. It is easy to see that the one-dimensional confidence interval will always underrepresented the true interval p_1 and p_2 are correlated.

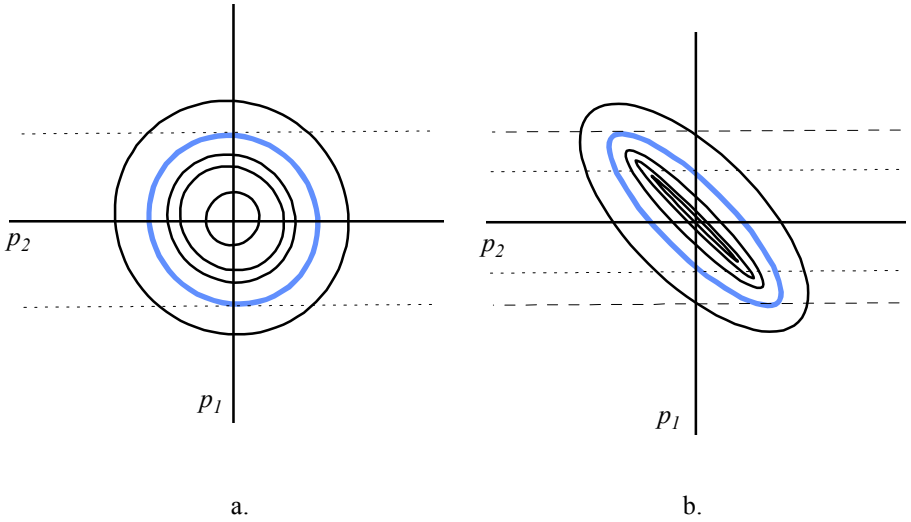

Figure 6 The log likelihood contour over the space of parameters p_1 and p_2. The bold ellipse represents the target change in likelihood that defines the upper and lower bounds of the confidence interval. Panel a: uncorrelated parameters, where the one dimensional change in likelihood is identical to the change over both parameters. Panel b: correlated parameters where the change in likelihood (dotted lines) is less than the change in likelihood over both parameters (dashed lines).

The confidence intervals are found iteratively in one dimension at a time. For each of the limit pairs, ***mle*** first evaluates the likelihood at the extremes `LOW` + `CI_LIMIT_DELTA` and `HIGH` + `CI_LIMIT_DELTA`. Convergence occurs when the difference between the likelihood at the parameter estimate and the confidence limit estimate is equal to `CI_CHISQ`, down to an absolute error of ±`CI_CONVERGE`. The maximum number of iterations for each of the limits is `CI_MAXITS`.

Printing distributions

The values of the survival function, the probability density function and the hazard function can be tabulated in the output by setting PRINT_DISTS = TRUE. All distributions that are in the model will be tabulated. The tabulation starts at value DIST_T_START, ends at the value DIST_T_END, and is tabulated for DIST_T_N equally spaced points. The mean value of data variables (e.g. covariates) are used when computing the distributions.

For example, to print the SDF, PDF, and hazard function at 100 points from 0 to 100 use the following code:

```
PRINT_DISTS   = TRUE     {print out distributions}
DIST_T_START  = 0        {lowest value to print}
DIST_T_END    = 100      {highest value to print}
DIST_T_N      = 101      {number of points to print}
```

Other printing options

The MIN_SIGNIFICANT variable controls the minimum number of significant digits in each numeric field of the confidence interval and standard error reports. More significant digits are displayed if there is room. If the number of leading zeros becomes too large, that number will be printed in scientific notation (1.2343E-56).

The variable PRINT_INFO, when TRUE, directs ***mle*** to print basic information about the model, including the method being used, the maximum number of iterations, the maximum number of function evaluations, and the criterion for normal convergence.

The PRINT_FREE_PARAMS variables, when TRUE, directs ***mle*** to print a list of all free parameters and the attributes of those parameters.

The variable PRINT_LLIKS controls printing of the individual likelihoods in a model. When TRUE, the likelihood and frequency for each observation will be printed to the output file.

Integration methods

Numeric integration can be difficult and slow. Furthermore there is no best method for integrating all functions. ***mle*** provides you with a number of different integration methods and several options for controlling those methods.

The method of integration is set using the variable INTEGRATE_METHOD. Currently there are four integration methods that are selected by assigning INTEGRATE_METHOD one of the following: I_AQUAD, I_SIMPSON, I_TRAP_OPEN, and I_TRAP_CLOSE.

INTEGRATE_METHOD = I_AQUAD selects an eight-point adaptive quadrature integration routine based on the Quanc8 routine of Forsythe et al (1977). This is probably the best all-around integration method included in ***mle***. With it, you can specify an tolerance to which integration will be done using the variable INTEGRATE_TOL variable or by including a third argument within the parenthesis of the INTEGRATE function call. When likelihoods include numerical integration, you should ensure that the integration tolerance is one or more magnitudes greater then the tolerance for the maximizer.

INTEGRATE_METHOD = I_SIMPSON selects a method of integration that is an extended Simpson's rule. When this method is selected, the function is split into smaller and smaller fractions until INTEGRATE_N levels have been evaluated. The subdivision will stop after INTEGRATE_N steps or when the relative error is less than INTEGRATE_TOL.

INTEGRATE_METHOD = I_TRAP_OPEN or INTEGRATE_METHOD = I_TRAP_CLOSE selects an integration routine which finds the integral using a trapezoidal approximation to the integral of INTEGRATE_N steps. When this method is selected, the number of subdivisions of the function will be INTEGRATE_N, and there is no convergence criterion. I_TRAP_OPEN uses an open extended Simpson's rule, so that the endpoints are never evaluated. The formula is

$$\int_a^b f(t)dt \approx \frac{b-a}{N}\left\{\begin{array}{l}\frac{109}{48}\left[f(t_2)+f(t_{N-1})\right]-\frac{5}{48}\left[f(t_3)+f(t_{N-2})\right] \\ +\frac{63}{48}\left[f(t_4)+f(t_{N-3})\right]+\frac{49}{48}\left[f(t_5)+f(t_{N-4})\right]+\sum_{i=6}^{N-5} f(t_i)\end{array}\right\},$$

where N is the number of subdivisions (INTEGRATE_N), and $t_i = a + (i–1)(b – a)/N$. I_TRAP_CLOSE uses a closed extended Simpson's rule, so that the endpoints are evaluated. The formula is

$$\int_a^b f(t)dt \approx \frac{b-a}{N}\left\{\begin{array}{l}\frac{17}{48}\left[f(t_1)+f(t_N)\right]-\frac{59}{48}\left[f(t_2)+f(t_{N-1})\right] \\ +\frac{43}{48}\left[f(t_3)+f(t_{N-2})\right]+\frac{49}{48}\left[f(t_4)+f(t_{N-3})\right]+\sum_{i=5}^{N-4} f(t_i)\end{array}\right\},$$

The error for both approximations is on the order N^{-4}.

Logistic equations

Logistic transformations are sometimes useful to change a variable in the range [-∞, ∞] to a transformed variable in the range [0, 1]. There are two common ways to construct these transformations. The transformation is $p_1 = 1/(1 + e^t)$, and the second transformation is $p_2 = e^t/(1 + e^t)$. Both transformation are equally useful, and there is no reason except habit to choose one over the other. Mathematically, they are related as complements; that is, $p_1 = 1 – p_2$.

By default, ***mle*** uses the transformation to p_1 for both the LOGISTIC() function call, and the FORM = LOGISTIC parameter transformation. You can change ***mle*** to use the p_2 form of the equation by setting the variable ALT_LOGISTIC = TRUE. The default value of ALT_LOGISTIC is false.

The interactive debugger

mle incorporates an interactive debugger that provides some degree of control while models are being solved. Entries in the symbol table can be viewed and changed, so that convergence can be forced early or postponed, output variables can be changed, and the values of various debugging options can be set and reset.

The debugger is called by typing <CTRL> C on most systems. The <BREAK> key also works on some systems. After ***mle*** gets to some reasonable stopping point—usually the end of an iteration—control will be passed to the user. The debugger responds with

Exit: immediately exits the program.

Resume: resumes running ***mle*** from where it left off.

One step: continue from where it left off for one more iteration and then reenters the interactive debugger.

Pick a symbol: selects a symbol to display. The value of the symbol is displayed between debugger commands, for this and all subsequent calls to the debugger.

Change the value of a symbol: If no symbol is selected, the user will be prompted for a symbol to change and then a value to change it to. If a symbol is selected (with ***Pick***), then that symbol will be changed.

Search for symbols: Prompts the user for search text, and then searches the symbol table for symbol names that match any part of the search text. The name, types, and value of matching symbols are displayed.

Predefined variables and constants

There are a plethora of pre-declared variables that are used to change the behavior of the program. These variables can occur anywhere in the main body of the program but not within a `MODEL...END` or a `DATA...END` statement.

The form used to assign a value to a variables is:

```
variable_name = <expression>
```

where <*expression*> can be an integer, a real, character, string or boolean (`TRUE` or `FALSE`) expression. (Expressions are discussed in the chapter on the Model statement). Certain of these types cannot be mixed. For instance, a string expression cannot be assigned to a variable already defined as a integer, real, or boolean. Likewise, a variable that is already defined as an integer cannot have a real number assigned to it subsequently. A variable that is type real can have an integer assigned to it; the integer will be converted to a real number first.

Table 15. Pre-defined variables.

Variable name	Default value	Comments
`ALT_LOGISTIC`	FALSE	Controls how logistic transformation are done in function and parameter form `LOGISTIC`. If set to false, they return $p=1/[1+\exp(x)]$. If true, the transformation is $p=\exp(x)/[1+\exp(x)]$.
`ANNEALING`	ANNEALING	A string constant for the simulated annealing method.
`ATOMICMASSU`	1.6606E-27	Atomic mass unit constant
`AVOGADROSN`	6.0220E-23	Avogadro's number
`BOHRMAGNETON`	9.2741E-24	Bohr's magneton
`BOHRRADIUS`	5.292E-11	Bohr's radius
`BRENT_ITS`	200	Defines the maximum number if iterations allowed for a single dimensional maximization when using the `BRENT` one-dimensional maximizer in the `DIRECT` and `CGRADIENT` methods.
`BRENT_MAGIC`	0.381966	[3 – sqrt(5)]/2, a constant used by the one-dimensional maximizer
`CGRADIENT1`	CGRADIENT1	A `METHOD` string so that the congugate gradient method (type 1) is used
`CGRADIENT2`	CGRADIENT2	A `METHOD` string so that the congugate gradient method (type 2) is used
`CI_CHISQ`	5.02389	Chi square value for likelihood CIs. This value defines a 95% CI.

CI_CONVERGE	0.00005	The maximum allowable difference between the likelihood at the confidence limit and CHISQ_C
CI_EVALS	0	Used to record the number of function evaluations when computing confidence intervals.
CI_LIMIT_DELTA	0.0	When confidence intervals are computed, ***mle*** adds this value to the lower parameter value and subtracts this parameter from the upper parameter value, thus preventing the confidence interval from being evaluated at the boundary. Negative values will extend the range over which the CI will be evaluated.
CI_MAXITS	30	The maximum number of iterations allowed for finding a confidence limit.
CONVERGENCE	1	Used to record the reason for terminating a solution
CREATE_OBS	0	For positive values, creates that number of observations rather than reading them from a data file; otherwise, observations are read from a file.
DATAFILE		The name of the file to be read in by a DATA statement. This variable is set using the DATAFILE() procedure.
DEBUG	0	A integer debug level. The higher this value, the more trees that die. Values of 5 and 11 are useful. (also -d ## on the command line)
DEBUG_DATA	FALSE	When TRUE, debugging is turned on for the routines that read in data files (also -dd on the command line).
DEBUG_INT	FALSE	When TRUE, turns on debugging for the integration routines (also -di on the command line).
DEBUG_LIK	FALSE	When TRUE, turns on debugging for likelihoods, and individual likelihoods are printed (also -dl on the command line).
DEBUG_PARSE	FALSE	When TRUE, turns on debugging for the language parsing routines (also -dp on the command line)
DEBUG_SYM	FALSE	When TRUE, turns on debugging for the symbol table routines (also -ds on the command line).
DEGREESPERRADIAN	57.2957795	The number of degrees in one radian = $\pi/180$
DELIMITERS	" , <tab>"	A string of delimiters that will define fields within a data file.
DELTA_LL	∞	Used to record the change in likelihood when solving models
DIFF_DX	0.001	The initial (largest) value of *dx* used for the DERIVATIVE function.
DIRECT	"DIRECT"	A METHOD string so that likelihoods are solved by the direct method
DIST_DX_SCALE	0.25	Multiplied by the standard error of a parameter when computing the derivative of distributions with respect to each parameter.
DIST_T_END	10.00	The default end time when distributions are printed. (See PRINT_DISTS and DIST_T_START)
DIST_T_N	10	The number of time points (between DIST_T_START and DIST_T_END) for which the distributions are printed. (See PRINT_DISTS).
DIST_T_START	1.00	The default start time when distributions are printed. (See PRINT_DISTS and DIST_T_END)
DX_MAXITS	12	The maximum number of iterations allowed for finding a reasonable sized Δx when computing numerical derivatives.
DX_START	0.1	The starting value for Δx used in finding numerical derivatives.
DX_TOOBIG	0.3	The maximum change in log likelihood allowed for computing a reasonable sized Δx.
DX_TOOSMALL	0.01	The minimum change in log likelihood allowed for computing a reasonable sized Δx
D_OBS		The number of data observations that were dropped.
E	2.71828...	The value *e*.
EPSILON	0.00001	The maximum change in log likelihood for convergence when estimating parameters for a model.
EULERSC	0.57721567	Euler's constant
EVALS	0	The number of function evaluations used to solve a model
EXP_HAZARD	True	Changes the way proportional hazards are modeled on the baseline hazard. if TRUE, $h(t) = h(t)'e^{p}$ otherwise $h(t) = h(t)'p$.
FALSE	FALSE	A boolean constant
FIND_EPS	0.00001	The convergence criterion for the FINDZERO and FINDMIN functions

FIND_MAXITER	100	Maximum number of iterations for the FINDZERO and FINDMIN functions
FREE_PARAMS	6.672E-11	The number of free parameters while solving a model
GRAVITATIONALC	6.672E-11	Universal gravitational constant
HIGH_DEFAULT	oo	The default value for the HIGH parameter limit. If no HIGH = ... is set for a parameter, the value will be taken from HIGH_DEFAULT.
INFINITY	(hardware)	A constant for the largest real number available on the computer. This number is determined when ***mle*** begins.
INFO_METHOD1	True	Computes the local Fisher's information matrix using Nelson's (1983:394) first derivative method. This method does not work for nested likelihoods, but is fairly robust for non-nested models.
INFO_METHOD2	False	When TRUE, computes the local Fisher's information matrix using numerical perterbation for the second partial derivatives.
INPUT_SKIP	0	Number of initial rows to skip when reading in data files. This is useful when, for example, columns of numbers in the input file have one or more lines of headings to describe the columns.
INTEGRATE_METHOD	I_AQUAD	Sets the method of integration.
INTEGRATE_N	15	Number of points in the Simpson and trapezoidal integrators.
INTEGRATE_TOL	0.001	The tolerance for the Simpson and adaptive quadrature integrators.
ITERATIONS	0	The number of iterations taken to solve a model.
ITERATION_PRINT	0	When set to *n*, every *n*th iteration will print out a partial result.
I_AQUAD	3	Constant for adaptive quadrature integrator (default)
I_SIMPSON	0	Constant for Simpson integrator
I_TRAP_CLOSED	1	Constant for trapezoidal (closed endpoint) integrator
I_TRAP_OPEN	2	Constant for trapezoidal (open endpoint) integrator
LARGEST_LIKELIHOOD	(hardware)	The largest likelihood acceptable from a function
LARGEST_LLIKELIHOOD	(hardware)	The largest loglikelihood acceptable from a function
LARGE_ZERO	(hardware)	A characteristic of the hardware floating point math.
LIGHTC	299792458	The speed of light
LINES_PER_OBS	1	The default number of lines per observation. This number is modified by ***mle*** if the LINE statement is used with the FIELD statement.
LINE_NUMB	0	Set to each data file line while reading in data. Afterward, it is set to the number of lines in the data file.
LNINFINITY	(hardware)	The log of the largest representable real number
LOGLIKELIHOOD	0.0	The loglikelihood found in solving the model
LOG_10	2.3025850	A constant.
LOW_DEFAULT	-oo	The default value for the LOW parameter limit. If no LOW = ... is set for a parameter, the value will be taken from LOW_DEFAULT.
MACHINE_EPSILON	(hardware)	Value associated with round-off error for the particular hardware being used. This number is determined when ***mle*** begins.
MAXEVALS	100000	The maximum number of function evaluations for solving a likelihood. Upon hitting MAXEVALS function evaluations, ***mle*** will terminate even if the convergence criterion has not been met.
MAXINT	(hardware)	Value of the greatest integer for the current architecture.
MAXITER	100	The maximum number of iterations allowed for estimating the parameters of a model. Upon hitting MAXITER iterations, ***mle*** will terminate even if the convergence criterion has not been met.
MAX_BOOLEANS	(hardware)	Fixed maximum size of a boolean array
MAX_CHARS	(hardware)	Fixed maximum size of a character array
MAX_INTEGERS	(hardware)	Fixed maximum size of an integer array
MAX_REALS	(hardware)	Fixed maximum size of a real array
MAX_STRINGS	(hardware)	Fixed maximum size of a string array
METHOD	DIRECT	METHOD takes on the value of one of several strings to define what method will be used for solving likelihoods.
METHOD_LOOP	FALSE	Turns on looping through methods until convergence is reached.
MINIMUM_ITS	1	The minimum number of iterations when solving a model.
MIN_SIGNIFICANT	4	The *minimum* number of significant digits to print for most fields in the parameter estimate reports.

NEGINFINITY	(hardware dependent)	The most negative real number supported by the machine. This value is determined when *mle* begins.
NEWTON	"NEWTON"	A METHOD used for finding parameter estimates.
N_OBS		The number of observations read and kept from the input file
N_VARS		The number of variables read and kept from the input file
NOTSINGULAR	FALSE	Returns the result of inverting the information matrix in computing the variance-covariance matrix. TRUE if the matrix is singular.
OO	(hardware)	The greatest positive real number. Also INFINITY
OUTFILE	(operating system dependent)	The name of the output file. Usually OUTFILE is defined in the *mle* program code using the procedure OUTFILE(). If OUTFILE is not defined in the program, the output will be sent to the standard output. OUTFILE can also be defined on the command line.
PARSE_ONLY	FALSE	When set to TRUE, *mle* parses the program file and then terminates. If syntax and other errors are encountered during parsing, *mle* will print the errors; otherwise, *mle* will simply terminate with error. The same effect is achieved by using -p on the *mle* command line.
PI	3.14159	The value π.
PLANCKINV2PI	1.0546E-34	Planck's constant divided by $2\times\pi$.
PLANCKSC	6.6262E-34	Planck's constant
POWELL	"POWELL"	A METHOD used for finding parameter estimates.
PRINT_BASIC	TRUE	Toggles printing of basic model information.
PRINT_CI	TRUE	Toggles printing of confidence interval report.
PRINT_COUNTS	TRUE	Toggles printing of variable counts from input variables.
PRINT_DATA_STATS	TRUE	Toggles printing of mean, standard deviation, minimum, maximum statistics for each input variable.

PRINT_DISTS	FALSE	Toggles printing of values for the survivorship, hazard, and PDF distributions at user-specified time points. When PRINT_DISTS is set to TRUE, values should also be set for DIST_T_START, DIST_T_END and DIST_T_N. The following code fragment will print the SDF, PDF, and hazard distributions at 100 time points from 100 to 300.
		`PRINT_DISTS = TRUE`
		`DIST_T_START = 100.0`
		`DIST_T_END = 300.0`
		`DIST_T_N = 100`

PRINT_FIELDS	FALSE	Toggles printing information about the field in the input file.
PRINT_FREE_PARAMS	FALSE	Toggles printing a list of free parameters sent to the maximizer. This is usually used for debugging purposes only.
PRINT_INFO	TRUE	Toggles printing of some information to the output file.
PRINT_LLIKS	TRUE	Toggles printing of individual likelihoods (1 per obs.) to the output file.
PRINT_OBS	FALSE	Toggles printing of the observations from the input file after transformations. When TRUE, prints the final values for all observations.
PRINT_SE	TRUE	Toggles printing of standard error report.
PRINT_SHORT	FALSE	When FALSE, *mle* prints a detailed reports for parameter estimates. When TRUE, *mle* prints a one-line report for the report. This option is useful when the results are to be manipulated directly by another program. The number of fields in the output report depend on how many parameters are estimated and whether the Standard Error report or the Confidence Limit report is generated.
PRINT_VCV	FALSE	Toggles printing of variance-covariance matrix. The rows and columns of the VCV matrix are in the same order as free parameters are defined.
PROGRAM_NAME	mle	
RADIANSPERDEGREE	0.01745329	The number of radians per degree = $\pi/180$
RANDOM_SEED	-1	The initial random seed. This must be set to a positive number (use the SEED() procedure) before using the random number generator. Note that the simulated annealing maximizer must have a random seed set.
RELEASE	–	The release number for *mle*.
REVISION	–	The revision number for *mle*.
RYDBERGC	0.01745329	Rydberg's constant

SA_ADJ_CYCLES	20	For simulated annealing, this is the number of step length adjustment steps every cooling cycle (iteration).
SA_ADJ_LOWERBOUND	0.4	For simulated annealing, picks the lower and upper percentages of accepted and rejected evaluations between which no step length adjustment is made.
SA_ALT_ADJUSTMENT	false	For simulated annealing, uses an alternative adjustment formula
SA_COOLING	0.85	This is the rate of cooling for each cooling cycle. $T_{n+1} = T_n \times SA\_COOLING$. Values > 1 can be used to explore a good starting temperature.
SA_EPS_NUMBER	4	For simulated annealing, this is the number of function points that will be compared for determining convergence.
SA_STEPLENGTH	1.0	For simulated annealing, this is the steplength constant.
SA_STEPLENGTH_ADJ	2.0	For simulated annealing, this is the steplength adjustment constant
SA_STEPS	5	For the simulated annealing method, this is the number of steps of random parameter perturbations before entering an adjustment cycle.
SA_TEMPERATURE	1000.0	For the simulated annealing method, this is the initial temperature. This value is conservatively high for most functions.
SIMPLEX	"SIMPLEX"	A METHOD string to denote Nelder and Mead's (1965) simplex maximizer.
SIMPLEX_ALPHA	1.0	The simplex maximizer's reflection coefficient
SIMPLEX_BETA	0.5	The simplex maximizer's contraction coefficient
SIMPLEX_GAMMA	2.0	The simplex maximizer's extrapolation coefficient
SMALLEST_LIKELIHOOD	(hardware)	The greatest value allowed for a likelihood
SMALLEST_ LLIKELIHOOD	(hardware)	The greatest value allowed for a log likelihood
SMALLEST_NUMBER	(hardware)	The smallest positive number greater than zero supported by the hardware.
SQRT_EPSILON	(hardware)	A small number used for computing derivatives.
START_DEFAULT	0.5	The default START value for parameters used in the event no START = ... is used in a parameter definition.
SURFACE_POINTS	20	Not yet used
SYM_TABLE_SIZE	401	An internal constant.
SYSTEM	–	Name of the operating system.
TEST_DEFAULT	0.0	The default *t*-test value against which the parameters are tested. This value is used when the TEST = is not used in a parameter definition.
TITLE		A string that is printed for the ***mle*** main program and each model. When TITLE is defined before a DATA statement, the title will be printed to the output as the global title. The title can be redefined before each MODEL statement as a model-specific statement.
TRUE	TRUE	A boolean constant.
UNIVERSALGASC	8.314410	The universal gas constant.
VCV_EVALS	0	The number of function evaluations in computing the variance-covariance matrix.
VCV_WIDTH	5	The number of elements printed on one line for the variance-covariance matrix.
VERBOSE	FALSE	If true, ***mle*** prints out status information as it works. This is useful for following the progress of ***mle***.
VERSION	–	The version number of ***mle*** .

PDFS AND THEIR CHARACTERISTICS

ARCSINE

This is the parameterless arcsine distribution. The arc sine distribution arises as a special case of the Beta distribution when $\nu = \omega$.

Parameters: none.

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $0 \leq t \leq 1$

PDF: $$f(t) = \frac{1}{\pi\sqrt{t(1-t)}}$$

SDF: $$S(t) = \arcsin\left(\sqrt{t}\right)\frac{2}{\pi}$$

Quantile: $$t_q = \sin\left(\frac{q\pi}{2}\right)^2$$

Mean: 1/2

Median: 1/2

Mode: 0 and 1

Variance: 1/8

References: Christensen (1984), Lévy (1939), Rao (1973)

See also: BETA

ASYMPTOTICRANGE

This is the asymptotic range distribution.

Parameters: a (location), b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $-\infty < t < \infty$

PDF: $$f(t) = \frac{2}{b} e^{-\frac{t-a}{b}} \mathrm{K}_0\left(2e^{-\frac{t-a}{2b}}\right)$$

SDF: $$S(t) = 1 - 2e^{-\frac{t-a}{2b}} \mathrm{K}_1\left(2e^{-\frac{t-a}{2b}}\right)$$

Mean: $a + 2\gamma b$

Median: $\approx a + 0.92860\ b$

Mode: $\approx 0.50637\ b$

Variance: $b^2\pi^2/3$

Constraints: $b \geq 0$

References: Christensen (1984), Gumbel (1947)

BERNOULLITRIAL

This distribution returns the probability from a single Bernoulli trial. The distribution has single intrinsic parameter (call it p) that is the probability of success. A single variable (call it *event*) is passed to the distribution and returns:

p, for *event* $\neq 0$

$1 - p$, for *event* $= 0$

Constraints: $0 \leq p \leq 1$

Mean: p

Variance: $p(1 - p)$

Notes: Since the Bernoulli distribution is not a time-based PDF, left- and right-truncation is not available. Covariate effects can be modeled on parameter p, but not on the hazard.

Example: code that determines fairness of coins from coin-tossing experiments is:

```
MODEL
        PDF BERNOULLITRIAL( is_heads )
                PARAM p  LOW = -999  HIGH = 999  START = 0  FORM = LOGISTIC
                  COVAR mint  PARAM b_mint  LOW = -5  HIGH = 5  START = 0
                  COVAR year  PARAM b_year  LOW = -5  HIGH = 5  START = 0
        END
END
```

See also: BINOMIAL

BETA

The Beta distribution is also called a Pearson Type I distribution. This distribution takes on values from 0 to 1. The distribution is J-shaped when $(\omega - 1)(\nu - 1) < 0$ and is U-shaped for $\omega < 1$ and $\nu < 1$. For other ν and ω the distribution is unimodal.

Parameters: ν (shape 1) and ω (shape 2)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $0 \le t \le 1$

PDF: $$f(t) = \frac{t^{\nu-1}(1-t)^{\omega-1}}{\mathrm{B}(\nu,\omega)}$$

SDF: $S(t) = \beta_p(\nu, \omega)$

Mean: $\nu/(\nu + \omega)$

Mode:
$$\begin{cases} (\nu-1)/(\nu+\omega-2), & \nu > 1,\ \omega > 1 \\ 0 & \nu < \omega,\ \nu \le 1 \text{ or } \omega \le 1 \\ 1 & \nu > \omega,\ \nu \le 1 \text{ or } \omega \le 1 \\ 0 \text{ and } 1 & \nu < 1,\ \omega < 1,\ \nu = \omega \\ 1/2 & \nu = 1,\ \omega = 1 \end{cases}$$

Variance: $\nu\omega/[(\nu + \omega)^2(\nu + \omega + 1)]$

Constraints: $\nu \ge 0$, $\omega \ge 0$, $\nu + \omega > 0$

Reduced models: Reduces to the arc sine distribution when $\nu = \omega$.

References: Bayes (1763), Christensen (1984), Rao (1973)

BINOMIAL

This is the binomial distribution with two parameters.

Parameters: p (proportion), n (count of Bernoulli trials).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$, t is an integer.

PDF: $$f(t) = \binom{n}{t} p^t (1-p)^{n-t}$$

SDF: $S(t) = B_p(t + 1, n - t)$

Mean: np

Mode: $p(n + 1)$

Variance: $np(1 - p)$

Constraints: $0 \leq p \leq 1$, $0 < n < \infty$, n is an integer.

References: Rao (1973)

See also: BERNOULLITRIAL

BIRNBAUMSAUNDERS

This is the Birnbaum-Saunders distribution.

Parameters: a (location), b (scale).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t > 0$

PDF: $$f(t) = \frac{1+\frac{a}{t}}{2b\sqrt{2\pi t}} \exp\left[\frac{-1}{2t}\left(\frac{t-a}{b}\right)^2\right]$$

SDF: $$S(t) = 1 - \Phi\left(\frac{t-a}{b\sqrt{t}}\right)$$

Quantile: $$t_q = \left(\frac{b}{2}\Phi_q + \sqrt{a + \frac{b^2}{2}\Phi_q}\right)^2$$

Mean: $a + b^2/2$

Median: a

Variance: $b^2\left(a + \frac{5}{4}b^2\right)$

Constraints: $a \geq 0$, $b \geq 0$

References: Birnbaum and Saunders (1969), Christensen (1984)

BIVNORMAL

This is the bivariate normal (or Gaussian) distribution with five intrinsic parameters.

Parameters: μ_x, σ_x, are the mean and standard deviation in the x dimension; μ_y, σ_y are the mean and standard deviations in the y dimension, and ρ is the correlation between X and Y.

Time variables: t_{ux}, t_{uy}, t_{ex}, t_{ey}, $t_{\alpha x}$, $t_{\alpha y}$, $t_{\omega x}$, $t_{\omega y}$.

Range: $-\infty < t < \infty$

PDF:
$$f(t_x, t_y) = \frac{\exp\left[-\frac{1}{2(1-\rho^2)}\left(\frac{(t-\mu_x)^2}{\sigma_x^2} - \frac{2\rho(t-\mu_x)(t-\mu_y)}{\sigma_x\sigma_y} + \frac{(t-\mu_y)^2}{\sigma_y^2}\right)\right]}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}}$$

Mean: μ_x, μ_y

Median: μ_x, μ_y

Mode: μ_x, μ_y

Variance: σ_x^2, σ_y^2

Covariance(X, Y): $\rho\sigma_x\sigma_y$

Constraints: $s_1 > 0$, $s_2 > 0$, $0 \le r \le 1$

Notes: Covariate effects cannot be modeled on the hazard.

See also: NORMAL

CAUCHY

This is the Cauchy distribution. The distribution is unimodal symmetric and with tails that extend to infinity. The quartiles are found as $a - b$ and $a + b$.

Parameters: a (location) and b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $-\infty < t < \infty$

PDF: $$f(t) = \frac{1}{\pi b\left[1 + \left(\frac{t-a}{b}\right)^2\right]}$$

SDF: $$S(t) = \frac{2\arctan\left(\frac{t-a}{b}\right) + \pi}{2\pi}$$

Quantile: $$t_q = a + b\tan\left[\pi\left(q - \tfrac{1}{2}\right)\right]$$

Mean: Doesn't exist

Median: a

Mode: a

Variance: ∞

Constraints: $b > 0$

References: Christensen (1984), Evans (1993), Rao (1973)

CHI

This is the three-parameter chi distribution.

Parameters: a (location), b (scale), c (shape).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{2^{1-\frac{c}{2}}}{b\Gamma(\frac{c}{2})}\left(\frac{t-a}{b}\right)^{c-1} e^{-\frac{1}{2}\left(\frac{t-a}{b}\right)^2}$$

SDF: $$S(t) = \frac{\Gamma\left[\frac{c}{2}, \frac{1}{2}\left(\frac{t-a}{b}\right)^2\right]}{\Gamma(\frac{c}{2})}$$

Hazard: $$h(t) = \frac{2^{1-\frac{c}{2}}\left(\frac{t-a}{b}\right)^{c-1} e^{-\frac{1}{2}\left(\frac{t-a}{b}\right)^2}}{b\Gamma(\frac{c}{2}) - b\gamma\left[\frac{c}{2}, \frac{1}{2}\left(\frac{t-a}{b}\right)^2\right]}$$

Quantile: $$q_t = a + b\sqrt{\chi_q^2(c)}$$

Mean: $$a + b\sqrt{2}\Gamma\left(\frac{c+1}{2}\right)\Gamma\left(\frac{c}{2}\right)^{-1}$$

Mode: $$\begin{cases} a + b\sqrt{c-1}, & c > 1 \\ a, & c \leq 1 \end{cases}$$

Variance: $$cb^2 - 2b^2\left[\Gamma\left(\frac{c+1}{2}\right)\Gamma\left(\frac{c}{2}\right)^{-1}\right]^2$$

Constraints: $a \geq 0$, $b > 0$, $c \geq 0$

Reduced models: Reduces to a Rayleigh distribution with $c = 2$, and a type of Maxwell distribution with $c = 3$.

References: Christensen (1984), Evans, et al. (1993)

See also: RAYLEIGH, CHISQUARED, MAXWELL

CHISQUARED

This is the central Chi-squared distribution.

Parameters: a (location), b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{t^{\frac{a}{2}-1} e^{-\frac{t}{2b}}}{(2b)^{\frac{a}{2}} \Gamma\left(\frac{a}{2}\right)}$$

SDF: $$S(t) = \frac{\gamma\left(\frac{a}{2}, \frac{t}{2b}\right)}{\Gamma\left(\frac{a}{2}\right)}$$

Hazard: $$h(t) = \frac{t^{\frac{a}{2}-1} e^{-\frac{t}{2b}}}{\gamma\left(\frac{a}{2}, \frac{t}{2b}\right)(2b)^{\frac{a}{2}}}$$

Quantile: $q_t = b\chi_q^2(a)$

Mean: ab

Median: $\approx ab - 2/3$, when ab is large.

Mode: $$\begin{cases} b(a-2), & a > 2 \\ 0, & a \leq 2 \end{cases}$$

Variance: $2ab^2$

Constraints: $a \geq 0$, $b > 0$

Reduced models: Reduces to an exponential distribution with $\lambda=(2b)^{-1}$ when $a = 2$.

References: Christensen (1984), Evans et al. (1993), Pearson (1900)

See also: GAMMA, CHI

COMPOUNDEXTREME

This is the compound extreme value distribution.

Parameters: a (location), b (scale), c (shape).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $-\infty < t < \infty$

PDF: $$f(t) = \frac{ca^c e^{\frac{t}{b}}}{\left(a + be^{\frac{t}{b}}\right)^{c+1}}$$

SDF: $$S(t) = \left(\frac{a}{a + be^{\frac{t}{b}}}\right)^c$$

Hazard: $$h(t) = \frac{c}{ae^{-\frac{t}{b}} + b}$$

Quantile $$t_q = b \ln\left\{\frac{a}{b}\left[(1-q)^{-c^{-1}} - 1\right]\right\}$$

Mean: $$b\left[\ln\left(\frac{a}{b}\right) + \gamma - \psi(c)\right]$$

Median: $$\ln\left[\frac{a}{b}\left(2^{\frac{1}{c}} - 1\right)\right] b$$

Mode: $$b \ln\left(\frac{a}{bc}\right)$$

Variance: $b^2[\psi'(1) - \psi'(c)]$

Constraints: $a > 0$, $b > 0$, $c > 0$

References: Christensen (1984)

See also: LARGEEXTREME

DANIELS

This is the parameterless Daniel's distribution.

Parameters: none

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $f(t) = (1+t)^{-2}$

SDF: $S(t) = (1+t)^{-1}$

Hazard: $h(t) = (1+t)^{-1}$

Quantile: $t_q = \frac{q}{1-q}$

Mean: ∞

Median: 1

Mode: 0

Variance: ∞

References: Christensen (1984), Daniels (1945)

DISK

The disk distribution begins at mode a and monotonically approaches zero at $a + 4b$, somewhat akin to a shifted negative exponential distribution.

Parameters: a (location), b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $a \le t \le a + 4b$

PDF:
$$f(t) = \frac{1}{b} - \frac{1}{\pi b}\sqrt{\frac{t-a}{b}\left(1 - \frac{t-a}{4b}\right)} - \frac{2}{\pi b}\arcsin\left(\frac{1}{2}\sqrt{\frac{t-a}{b}}\right)$$

SDF:
$$S(t) = \frac{3}{2} - \frac{t-a}{b} + \frac{t-a+2b}{4\pi b}\sqrt{\frac{t-a}{b}\left(4 - \frac{t-a}{b}\right)} + \frac{1}{\pi}\arcsin\left(\frac{t-a-2b}{2b}\right) + 2\frac{t-a-2b}{\pi b}\arcsin\left(\frac{1}{2}\sqrt{\frac{t-a}{b}}\right)$$

Mean: $a + b$

Median: $\approx a + 0.7944\, b$

Mode: a

Variance: $2b^2/3$

Constraints: $b > 0$

References: Borel (1925), Christensen (1984)

EXPONENTIAL

The exponential is commonly used in reliability engineering, queuing theory and biology. The 'memoryless' property of the exponential distribution is an important characteristic. It says, in effect, that for a survivor, future times to failure are completely independent of the past. Another way to express the property is that the hazard of failure is constant.

Parameter:	λ (hazard and 1/scale).
Time variables:	t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$
Range:	$t \geq 0$
PDF:	$f(t) = \lambda \exp(-\lambda t)$
SDF:	$S(t) = \exp(-\lambda t)$
Hazard:	$h(t) = \lambda$
Quantile:	$t_q = -\ln(1 - q)/\lambda$
Mean:	$1/\lambda$
Median:	$\ln(2)/\lambda$
Mode:	0
Variance:	$1/\lambda^2$
References:	Christensen (1984), Evans et al. (1993), Nelson (1982)
See also:	The SHIFTEXPONENTIAL distribution is a 2 parameter (location-scale) version of this distribution.

GAMMA

This is the gamma distribution, also known as the Pearson Type III distribution.

Parameters: λ (hazard and 1/scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{\lambda^c t^{c-1}}{\Gamma(c) e^{\lambda t}}$$

SDF: $$S(t) = \frac{\Gamma(c, \lambda t)}{\Gamma(c)}$$

Hazard: $$h(t) = \frac{\lambda^c t^{c-1}}{\Gamma(c, \lambda t) e^{\lambda t}}$$

Quantile: $$t_q = \frac{\chi^2_q(2c)}{2\lambda}$$

Mean: c/λ

Mode: $$\begin{cases} (c-1)/\lambda, & c > 1 \\ 0 & c \leq 1 \end{cases}$$

Variance: c/λ^2

Constraints: $\lambda > 0$, $c > 0$

Reduced models: Reduces to an exponential distribution when $c = 1$. Reduces to an Erland distribution when parameter c is an integer. This distribution is the SHIFTGAMMA distribution with $a = 0$ and $1/b = \lambda$.

References: Christensen (1984), Elandt-Johnson and Johnson (1980), Evans et al. (1993), Kalbfleisch and Prentice (1980).

See also: EXPONENTIAL, SHIFTGAMMA, GENGAMMA

GAMMAFRAIL

This model has a constant hazard for individuals, but gamma-distributed frailty (heterogeneity) among individuals. The model is used primarily because the PDF, SDF and hazard functions have simple forms.

Parameters: λ (hazard and 1/scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{\lambda c^{c+1}}{(\lambda t + c)^{c+1}}$$

SDF: $$S(t) = \left(\frac{c}{c + \lambda t} \right)^c$$

Hazard: $$h(t) = \frac{\lambda c}{ct + c}$$

Quantile: $$t_q = \frac{c}{\lambda} \left(\frac{1}{\sqrt[c]{q}} - 1 \right)$$

Mean: $$\frac{c}{\lambda(c-1)}$$

Variance: $$\frac{c^3}{\lambda^2 (c-1)^2 (c-2)}$$

Constraints: $\lambda > 0$, $c \geq 0$

References:

See also: EXPONENTIAL, GAMMA

GENGAMMA

This is the three parameter (shifted) gamma distribution, also known as the Pearson Type III distribution.

Parameters: a (location), b (scale, inverse of the hazard), c (1st shape), d (2nd shape).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq a$

PDF: $$f(t) = \frac{d(t-a)^{cd-1} e^{-\left(\frac{t-a}{b}\right)^d}}{b^{cd}\Gamma(c)}$$

SDF: $$S(t) = \frac{\Gamma\left[c, \left(\frac{t-a}{b}\right)^d\right]}{\Gamma(c)}$$

Hazard: $$h(t) = \frac{d(t-a)^{cd-1} e^{-\left(\frac{t-a}{b}\right)^d}}{b^{cd}\Gamma\left[c, \left(\frac{t-a}{b}\right)^d\right]}$$

Quantile: $$t_q = a + b\left[\tfrac{1}{2}\chi^2_q(2c)\right]^{-1}$$

Mean: $$a + b\frac{\Gamma\left(c + d^{-1}\right)}{\Gamma(c)}$$

Mode: $$\begin{cases} a + \frac{b}{d}\left(\frac{cd-1}{d}\right)^{d-1} & cd > 1 \\ a & cd \leq 1 \end{cases}$$

Variance: $b^2 c$

Constraints: $b > 0$, $c > 0$, $d > 0$

Reduced models: Reduces to the shifted gamma distribution when $d = 1$. Reduces to the shifted exponential when $c = 1$ and $d = 1$. Reduces to the shifted Weibull distribution when $c = 1$. Reduces to the Chi-squared distribution with v degrees of freedom when $a = 0$, $b = 2$, $c = v/2$, and $d = 1$.

References: Christensen (1984), Evans et al. (1993), Kalbfleisch and Prentice (1980).

See also: SHIFTEXPONENTIAL, GAMMA, SHIFTGAMMA, CHISQUARED, SHIFTWEIBULL

GENGUMBEL

This is the three-parameter generalized Gumbel distribution.

Parameters: a (location), b (scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $-\infty < t < \infty$

PDF: $$f(t) = \frac{a}{b\Gamma(c)} \exp\left[-\frac{ct}{b} - ae^{-\frac{t}{b}} \right]$$

SDF: $$S(t) = \frac{\gamma\left(c, ae^{-\frac{x}{b}}\right)}{\Gamma(c)}$$

Hazard: $$f(t) = \frac{a}{b\gamma\left(c, ae^{-\frac{x}{b}}\right)} \exp\left[-\frac{ct}{b} - ae^{-\frac{t}{b}} \right]$$

Mean: $b[\ln(a) - \psi(c)]$

Median: $b\ln(a/c)$

Variance: $b^2\psi'(c)$

Constraints: $a > 0$, $b > 0$, $c > 0$

References: Ahuja and Nash (1967), Christensen (1984)

See also: GUMBEL

GEOMETRIC

This is the discrete geometric distribution,. The distribution describes the times up to and including the first success in a sequence of Bernoulli trials. The geometric distribution is the discrete analogue of the negative exponential distribution.

Parameters:	p (probability)
Time variables:	t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.
Range:	$t \geq 1$, t is an integer
PDF:	$f(t) = p(1-p)^{t-1}$
SDF:	$S(t) = (1-p)^t$
Mean:	$1/p$
Mode:	1
Variance:	$(1-p)p^2$
Constraints:	$0 \leq p \leq 1$
References:	Evans et al. (1993)
See also:	BERNOULLITRIAL, EXPONENTIAL, NEGBINOMIAL, HYPERGEOMETRIC

GOMPERTZ

This is the Gompertz distribution, which is sometimes used as a model of senescent mortality (with $b > 0$) and infant mortality (with $b < 0$).

Parameters: a (scale), b (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = a \exp\left[bt + \frac{a}{b}(1 - e^{bt}) \right] N \qquad N = \begin{cases} 1, & b \geq 0 \\ \left(1 - e^{\frac{a}{b}}\right)^{-1} & b < 0 \end{cases}$$

SDF: $$S(t) = \exp\left[\frac{a}{b}(1 - e^{bt}) \right] N$$

Hazard: $h(t) = a \exp(bt)$

Constraints: $a \geq 0$

Reduced models: When $b = 0$, the PDF is exponential with parameter a.

References: Christensen (1984)

See also: EXPONENTIAL, MAKEHAM, MIXMAKEHAM, SILER

HORSESHOE

This is the horseshoe distribution. The distribution is a mirrored power function; it discontinuous at the mean, and the mirror-like symmetrical is down the mean. Except when c is zero, the distribution is always bimodal.

Parameters: a (location), b (scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $(a - b) \le t \le (a + b)$

PDF: $$f(t) = \frac{c+1}{2b}\left|\frac{x-a}{b}\right|^c$$

SDF: $$S(t) = \begin{cases} \frac{1}{2}\left[1-\left(\frac{x-a}{b}\right)^{c+1}\right], & x \ge a \\ \frac{1}{2}\left[1+\left(-\frac{x-a}{b}\right)^{c+1}\right], & x \le a \end{cases}$$

Quantile $$t_q = \begin{cases} a+b(2q-1)^{(c+1)^{-1}}, & q \ge 1/2 \\ a-b(1-2q)^{(c+1)^{-1}}, & q \le 1/2 \end{cases}$$

Mean: a

Median: a

Mode: $a \pm b$

Variance: $b^2(c + 1)/(c + 3)$

Constraints: $b > 0$, $c \ge 0$

Reduced models: Reduces to a uniform distribution when $c = 0$. Reduces to the symmetric quad when $c = 2$, the symmetric quart when c is 4, and the symmetric sextic when $c = 6$.

References: Christensen (1985)

See also: POWER

HYPERBOLICSECANT

This is the hyperbolic secant distribution. The distribution is symmetric about a.

Parameters: a (location), b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{1}{\pi b}\operatorname{sech}\left(\frac{t-a}{b}\right)$$

SDF: $$S(t) = 1 - \frac{2}{\pi}\arctan\left[\exp\left(\frac{t-a}{b}\right)\right]$$

Quantile: $$t_q = a + b\ln\left[\tan\left(\frac{\pi q}{2}\right)\right]$$

Mean: a

Median: a

Mode: a

Variance: $b^2\pi^2/4$

Constraints: $b > 0$

References: Christensen (1984), Perks (1932), Talacko (1956)

HYPERGEOMETRIC

This is the hypergeometric distribution.

Parameters: p (probability), m, n

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: integer $t \geq 0$

PDF: $$f(t) = \frac{\binom{np}{t}\binom{n-np}{m-t}}{\binom{n}{m}}$$

SDF: $$S(t) = \sum_{i=0}^{t} \frac{\binom{np}{i}\binom{n-np}{m-i}}{\binom{n}{m}}$$

Mean: mp

Mode: $$\frac{(np+1)(m+1)}{n+2}$$

Variance: $$\frac{mp(p+1)(n-m)}{n-1}$$

Constraints: $t \geq 0$, $m - n + np \leq t \leq \min(m, np)$, $0 \leq p \leq 1$, $1 \leq m \leq n$, np is an integer.

References: Laplace (1774)

See also: GEOMETRIC

HYPER2EXP

This is a two-point hyperexponential distribution, also called a mixed exponential distribution. It arises when the entire system fails when either of two constant-hazard components fails. The distribution has been used previously as a model of fetal loss (Wood 1994; Holman 1996) and CPU service times (Kishor et al. 1982).

Parameters: p (initial proportion in subgroup 1); λ_1 (constant hazard in subgroup1); λ_2 (constant hazard in subgroup 2).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $f(t) = p\lambda_1 \exp(-\lambda_1 t) + (1-p)\,\lambda_2 \exp(-\lambda_2 t)$

SDF: $S(t) = p\,\exp(-\lambda_1 t) + (1-p)\,\exp(-\lambda_2 t)$

Hazard: $$h(t) = \frac{p\lambda_1 \exp(-\lambda_1 t) + (1-p)\lambda_2 \exp(-\lambda_2 t)}{p\exp(-\lambda_1 t) + (1-p)\exp(-\lambda_2 t)}$$

Mean: $$\frac{p}{\lambda_1} + \frac{(1-p)}{\lambda_2}$$

Mode: 0

Variance: $$\frac{2p}{\lambda_1^2} + \frac{2(1-p)}{\lambda_2^2} - \left[\frac{p}{\lambda_1} + \frac{(1-p)}{\lambda_2}\right]^2$$

Constraints: $0 \leq p \leq 1$; $\lambda_1 \geq 0$; $\lambda_2 \geq 0$

Reduced models: When $\lambda_1 = \lambda_2$ and p is fixed to any value between 0 and 1, the PDF is exponential with parameter λ_1.

References: Christensen (1984), Holman (1995), Kishor (1982)

See also: EXPONENTIAL, HYPO2EXP

HYPO2EXP

This is a 2-point hypoexponential distribution. It describes a two stage process in which two independent exponentially distributed components must both fail for the entire system to fail. It arises by taking the convolution of two independent and exponentially distributed components. The distribution has been used to describe I/O operations in computer systems (Kishor et al. 1982).

Parameter: λ_1 (hazard for the first component), λ_2 (hazard for the second component)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{\lambda_1\lambda_2\left[\exp(-\lambda_1 t) - \exp(-\lambda_2 t)\right]}{\lambda_2 - \lambda_1}$$

SDF: $$S(t) = \frac{\lambda_2 \exp(-\lambda_1 t) - \lambda_1 \exp(-\lambda_2 t)}{\lambda_2 - \lambda_1}$$

Hazard: $$h(t) = \frac{\lambda_1\lambda_2\left[\exp(-\lambda_1 t) - \exp(-\lambda_2 t)\right]}{\lambda_2 \exp(-\lambda_1 t) - \lambda_1 \exp(-\lambda_2 t)}$$

Mean: $$\frac{1}{\lambda_1} + \frac{1}{\lambda_2}$$

Variance: $$\frac{1}{\lambda_1^2} + \frac{1}{\lambda_2^2}$$

Constraints: $\lambda_1 \geq 0$; $\lambda_2 \geq 0$

References: Christensen (1984), Kishor (1982)

See also: EXPONENTIAL, HYPER2EXP

INVBETA1

This is the inverted beta type 1 distribution.

Parameters: a, b, c

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq c$

PDF: $$f(t) = \frac{c^a (t-c)^{b-1}}{\mathrm{B}(a,b)t^{a+b}}$$

SDF: $S(t) = \mathrm{B}_{c/t}(a, b)$

Mean: $$\frac{c(a+b-1)}{a-1} \qquad a > 1$$

Mode: $$\begin{cases} \dfrac{c(a+b)}{a+1} & b > 1 \\ c & b \leq 1 \end{cases}$$

Variance: $$\frac{bc^2(a+b-1)}{(a-1)^2(a-2)} \qquad a > 2$$

Constraints: $a \geq 0$, $b > 0$, $c \geq 0$

References: Christensen (1984)

See also: INVBETA2

INVBETA2

This is the inverted beta type 2 distribution.

Parameters: a, b, c

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{c^b t^{a-1}}{\mathrm{B}(a,b)(t+c)^{a+b}}$$

SDF: $S(t) = \beta_{t/(t+c)}(a, b)$

Mean: $$\frac{ca}{b-1} \qquad b > 1$$

Mode: $$\begin{cases} \dfrac{c(a-1)}{b+1} & a > 1 \\ 0 & a \leq 1 \end{cases}$$

Variance: $$\frac{ac^2(a+b-1)}{(b-1)^2(b-2)} \qquad b > 2$$

Constraints: $a > 0$, $b > 0$, $c \geq 0$

References: Christensen (1984)

See also: INVBETA1

INVCHI

This is the xxx distribution.

Parameters: .

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{e^{-\frac{b^2}{2t^2}} 2^{1-a/2}}{b\Gamma(a/2)} \left(\frac{b}{t}\right)^{a+1}$$

SDF: $$S(t) = \frac{\gamma\left[a/2, \left(\frac{b}{4t}\right)^2\right]}{\Gamma(a/2)}$$

Mean: $$S(t) = \frac{b\Gamma\left[\frac{a-1}{2}\right]}{\sqrt{2}\Gamma(a/2)} \qquad a > 1$$

Mode: $$\frac{b}{\sqrt{a+1}}$$

Variance: $$S(t) = \frac{b^2}{a-2} - \frac{b^2\Gamma\left[\frac{a-1}{2}\right]^2}{2\Gamma(a/2)^2}, \qquad a > 2$$

Constraints: $a > 0$, $b > 0$

References: Christensen (1984)

See also: CHI

INVGAMMA

This is the inverted gamma distribution, also called the Person type V distribution.

Parameters: b (scale), c (shape).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{e^{-b/t}}{b\Gamma(c)}\left(\frac{b}{t}\right)^{c+1}$$

SDF: $$S(t) = \frac{\gamma(c, b/t)}{\Gamma(c)}$$

Hazard: $$h(t) = \frac{e^{-b/t}}{b\gamma(c, b/t)}\left(\frac{b}{t}\right)^{c+1}$$

Quantile: $$t_q = \frac{2b}{\chi^2_{1-q}(2c)}$$

Mean: $b / (c - 1)$, $a > 1$

Mode: $b / (c + 1)$

Variance: $b^2(c-1)^{-2}(c-2)^{-1}$, $a > 2$

Constraints: $a > 0$, $b > 0$

References: Christensen (1984), Evans et al. (1993), Pearson (1895)

See also: GAMMA, INVBETA1, INVBETA2

INVGAUSSIAN

This is the inverse Gaussian distribution, which includes the Wald distribution as a special case.

Parameters: b (scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t > 0$

PDF: $$f(t) = \sqrt{\frac{b}{2\pi t^3}} \exp\left[-\frac{b}{2c^2 t}\left(\frac{t}{b} - c\right)^2 \right]$$

SDF: $$S(t) = 1 - \Phi\left(\frac{t - bc}{c\sqrt{bt}}\right) - e^{2/c}\Phi\left(\frac{-t - bc}{c\sqrt{bt}}\right)$$

Mean: bc

Mode: $$bc\left(\sqrt{1 + 9c^2/4} + 3c/2\right)$$

Variance: b^2c^3

Constraints: $b \geq 0$, $c \geq 0$

Reduced models: Approaches the Normal distribution as $b \to \infty$. Reduces to the Wald distribution when $b = 1/c$.

References: Christensen (1984), Evans et al. (1993), Schrödinger (1915), Tweedie (1947), Wald (1947)

See also: RANDOMWALK

LAPLACE

This is the Laplace distribution, also known as a double-exponential distribution. The distribution is discontinuous at a, and declines to the left and right exponentially.

Parameters: a (location), b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{e^{-\left|\frac{t-a}{b}\right|}}{2b}$$

SDF: $$S(t) = \begin{cases} \frac{1}{2} e^{-\frac{t-a}{b}}, & t \geq a \\ 1 - \frac{1}{2} e^{-\frac{a-t}{b}}, & t < a \end{cases}$$

Mean: a

Median: a

Mode: a

Variance: $2b^2$

Constraints: $b \geq 0$

References: Christensen (1984), Evans et al. (1993), Laplace (1774)

See also: EXPONENTIAL, SUBBOTIN

LARGEEXTREME and GUMBEL

Parameters: a (location) and b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $-\infty < t < \infty$

PDF: $$f(t) = \frac{1}{b}\exp\left[-\frac{t-a}{b} - \exp\left(-\frac{t-a}{b}\right)\right]$$

SDF: $$S(t) = 1 - \exp\left\{-\exp\left(-\frac{t-a}{b}\right)\right\}$$

Hazard: $$h(t) = \frac{\exp\left(-\frac{t-a}{b}\right)}{b\left[\exp\left[-\exp\left(-\frac{t-a}{b}\right)\right]-1\right]}$$

Mean: $a - kb$ $k \approx 0.57721...$ is Euler's constant

Median: $a - b\log[\log(2)]$

Mode: a (≈36.8th percentile)

Variance: $b^2\pi^2/6$

Constraints: $b \geq 0$

References:

LINEARHAZARD

The linear hazard rate distribution is so-called because the hazard increases monotonically with time.

Parameters: λ (constant hazard), b (scale).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $f(t) = (\lambda + bt)\exp(-\lambda t - \frac{1}{2}bt^2)$

SDF: $S(t) = \exp(-\lambda t - \frac{1}{2}bt^2)$

Hazard: $h(t) = \lambda + bt$

Constraints: $\lambda \geq 0$, $b > 0$

References: Lee (1992)

LNGAMMA

This is the log-gamma distribution.

Parameters: a (location), b (scale, inverse of the hazard), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq a$

PDF:
$$f(t) = \frac{\left(\frac{\ln(t)-a}{b}\right)^{c-1} e^{-\frac{\ln(t)-a}{b}}}{b\Gamma(c)}$$

SDF:
$$S(t) = \frac{\Gamma\left(c, \frac{\ln(t)-a}{b}\right)}{\Gamma(c)}$$

Hazard:
$$h(t) = \frac{\left(\frac{\ln(t)-a}{b}\right)^{c-1} e^{-\frac{\ln(t)-a}{b}}}{b\Gamma\left(c, \frac{\ln(t)-a}{b}\right)}$$

Constraints: $b \geq 0$, $c \geq 0$

References:

See also: GAMMA

LNLOGISTIC

This is the log-logistic distribution.

Parameters: a (location), b (scale).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$

Range: $-\infty < t < \infty$

PDF: $$f(t) = \frac{\frac{e^{-a}}{b}\left[e^{-a}t\right]^{\frac{1}{b}-1}}{\left[1+(e^{-a}t)^{\frac{1}{b}}\right]^2}$$

SDF: $$S(t) = \left[1+e^{-a}t\right]^{-\frac{1}{b}}$$

Hazard: $$h(t) = \frac{\frac{e^{-a}}{b}\left[e^{-a}t\right]^{\frac{1}{b}-1}}{1+(e^{-a}t)^{\frac{1}{b}}}$$

Constraint: $b > 0$

References: ?

See also: LOGISTIC

LOGISTIC

The logistic distribution is also called the sech-squared distribution.

Parameters: a (location), b (scale).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$

Range: $-\infty < t < \infty$

PDF:
$$f(t) = \frac{\exp\left(-\frac{t-a}{b}\right)}{b\left[1+\exp\left(-\frac{t-a}{b}\right)\right]^2}$$
$$= \frac{1}{4b}\operatorname{sech}^2\left(\frac{t-a}{2b}\right)$$

SDF:
$$S(t) = \left\{1+\exp\left[\frac{t-a}{b}\right]\right\}^{-1}$$
$$= \frac{1}{2}\left[1-\tanh\left(\frac{t-a}{2b}\right)\right]$$

Hazard:
$$h(t) = \frac{1}{b}\left\{1+\exp\left[-\frac{t-a}{b}\right]\right\}^{-1}$$

Quantile:
$$t_q = a - b\ln\left(\frac{1-q}{q}\right)$$

Mean: a

Median: a

Mode: a

Variance: $\pi^2 b^2/3$

Constraint: $b > 0$

References: Christensen (1984), Evans et al. (1993)

See also: LNLOGISTIC

LOGNORMAL or LNNORMAL

This is the lognormal distribution, sometimes called the Cobb-Douglas, the Galton-McAlister, or the Kapteyn-Gibrat distributions.

Parameters: a (location), b (scale).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$

Range: $t \geq a$

PDF: $$f(t) = \frac{1}{tb\sqrt{2\pi}} e^{\left[\frac{-[\ln(t-a)]^2}{2b^2}\right]}$$

SDF: $$S(t) = 1 - \Phi\left[\frac{\ln(t-a)}{b}\right]$$

Mean: $a\exp(b^2/2)$

Median: a

Mode: $a/\exp(b^2)$

Variance: $a^2\exp(b^2/2)[\exp(b^2/2) - 1]$

Constraints: $b > 0$

Notes: This distribution is related to the SHIFTLOGNORMAL distribution as follows. The two-parameter LNNORMAL cumulative density is found from the Normal density by taking $\Phi[\ln(t - a)/b]$ whereas in the three parameter SHIFTLOGNORMAL we take $\Phi\{\ln[(t - a)/b]/c\}$.

References: Evans et al. (1993), Nelson (1982)

See also: NORMAL, SHIFTLOGNORMAL

LOGSERIES

This is the logarithmic series distribution. This distribution can be derived from both a power series distribution and a negative binomial distribution.

Parameters: p (probability)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: integer $t \geq 1$

PDF: $$f(t) = -\frac{p^t}{t \ln(1-p)}$$

SDF: $$S(t) = 1 + \frac{1}{\ln(1-p)} \sum_{j=1}^{t} \frac{p^t}{t}$$

Mean: $$-\frac{p}{\ln(1-p)(1-p)}$$

Mode: 1

Variance: $$-\frac{p\left(1 + \frac{p}{\ln(1-p)}\right)}{\ln(1-p)(1-p)^2}$$

Constraints: $0 \leq p \leq 1$

References: Christensen (1984), Evans et al. (1993), Fisher (1943)

See also: POWERSERIES, NEGBINOMIAL

LOWMAX

This is the Lowmax distribution, which is a generalized Pareto distribution.

Parameters: a (location), b (scale), c (shape).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $a + b \le t < \infty$

PDF: $$f(t) = \frac{b^c c}{(t-a)^{c+1}}$$

SDF: $S(t) = [(t-a)/b]^{-c}$

Quantile: $t_q = a + b(1-q)^{-1/c}$

Mean: $a + bc/(c-1), \qquad c > 1$

Median: $a + b\sqrt[c]{2}$

Mode: $a + b$

Variance: $$\frac{b^2 c}{(c-2)(c-1)^2}, \quad c > 2$$

Constraints: $b \ge 0$, $c > 0$

Reduced models: Reduces to the Pareto distribution when $a = 0$;

References: Christensen (1984)

See also: PARETO

MAKEHAM

This is the Makeham-Gompertz (also called the Gompertz-Makeham) distribution frequently used as a competing hazards model for mortality. The a_1 parameter is interpreted as accidental mortality component, and the a_2 and b parameters make up the senescent mortality component.

Parameters: a_1, a_2 and b

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \left(a_1 + a_2 e^{bt}\right) \exp\left[-a_1 t + \frac{a_2}{b}(1 - e^{bt})\right],$$

SDF: $$S(t) = \exp\left[-a_1 t + \frac{a_2}{b}(1 - e^{bt})\right]$$

Hazard: $h(t) = a_1 + a_2 \exp(bt)$

Constraints: $a_1 \geq 0$, $a_2 \geq 0$

Reduced models: $a_1 = 0$ reduces to a Gompertz PDF with parameters a_2 and b. $b = 0$ and either a_1 or a_2 are constrained, reduces to an exponential with parameter $a_1 + a_2$.

References: Elandt-Johnson and Johnson (1980)

See also: EXPONENTIAL, GOMPERTZ, MIXMAKEHAM, SILER

MAXWELL

This is the two-parameter Maxwell-Boltzmann distribution. It is used to model the distribution of particles at equilibrium in statistical mechanics.

Parameters: a (location), b (scale).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \frac{4}{b\sqrt{\pi}}\left(\frac{t-a}{b}\right)^2 e^{-\left(\frac{t-a}{b}\right)^2},$$

SDF: $$S(t) = \operatorname{erf}\left(\frac{t-a}{b}\right) - \frac{2}{\sqrt{\pi}}\left(\frac{t-a}{b}\right)e^{-\left(\frac{t-a}{b}\right)^2},$$

Mean: $$a + \frac{2b}{\sqrt{\pi}}$$

Mode: $a + b$

Variance: $b^2\left(\frac{3}{2} - \frac{4}{\pi}\right)$

Constraints: $b \geq 0$

References: Christensen (1984), Maxwell (1860a,b), Rao (1973)

See also: RAYLEIGH, CHISQUARED, MAXWELL

MIXMAKEHAM

This is the mixed-Makeham distribution, which can be used to model the human lifespan

Parameters: p (initial proportion in risk group 1), λ_1 (constant hazard in subgroup 1), λ_2 (constant hazard in subgroup 2), λ_3 (senescent hazard), b (senescent shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF:
$$f(t) = Np\exp\left[-\lambda_1 t + \frac{\lambda_3}{b}\left(1-e^{bt}\right)\right]\left(\lambda_1 + \lambda_3 e^{bt}\right)$$
$$+N\left[1-p\right]\exp\left[-\lambda_2 t + \frac{\lambda_3}{b}\left(1-e^{bt}\right)\right]\left(\lambda_2 + \lambda_3 e^{bt}\right),$$
$$N = \begin{cases} 1, & b \geq 0 \\ \left(1-e^{\frac{\lambda_3}{b}}\right)^{-1} & b < 0 \end{cases}$$

SDF:
$$S(t) = Np\exp\left[-\lambda_1 t + \frac{\lambda_3}{b}\left(1-e^{bt}\right)\right] + N\left[1-p\right]\exp\left[-\lambda_2 t + \frac{\lambda_3}{b}\left(1-e^{bt}\right)\right]$$

Hazard:
$$h(t) = p(t)\lambda_1 + [1-p(t)]\lambda_2 + \lambda_3 e^{bt}$$

where
$$p(t) = \frac{p\exp\left[-\lambda_1 t + \frac{\lambda_3}{b}\left(1-e^{bt}\right)\right]}{p\exp\left[-\lambda_1 t + \frac{\lambda_3}{b}\left(1-e^{bt}\right)\right] + \left[1-p\right]\exp\left[-\lambda_2 t + \frac{\lambda_3}{b}\left(1-e^{bt}\right)\right]}$$

Constraints: $0 \leq p \leq 1$; $\lambda_1 \geq 0$, $\lambda_2 \geq 0$, $\lambda_3 \geq 0$

Reduced models: Reduces to a Makeham-Gompertz if p = 0 or 1, reduces to an 2 point hyperexponential exponential if $\lambda_3 = 0$, reduces to an exponential if $\lambda_3 = 0$ and p = 0 or 1, reduces to a Gompertz if $\lambda_1 = 0$ and $\lambda_2 = 0$.

References:

See also: MAKEHAM, GOMPERTZ, SILER, EXPONENTIAL

NEGBINOMIAL

This is the negative binomial distribution, also known as the Pascal distribution when t is an integer.

Parameters: p (probability), n (count).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $f(t) = \binom{t+n-1}{n-1} p^n (1-p)^t$ (this is wrong--need continuous version)

SDF: $S(t) = \beta_p(n, t+1)$

Mean: $n(1-p)/p$

Mode: Floor$[p(n-1)/(1-p)]$

Variance: $n(1-p)/p^2$

Constraints: $0 \leq p \leq 1$, $n > 0$

Reduced models: Reduces to a Pascal distribution when t is an integer. Reduces to the geometric distribution when $n = 1$.

References: Christensen (1984), Evans et al. (1993)

See also: GEOMETRIC, POWERSERIES, BINOMIAL

NORMAL or GAUSSIAN

This is the commonly-used normal distribution, also called the Gaussian distribution and Laplace's second law of error. The normal is odd (but certainly possible) as a failure time distribution because times can take any value from -∞ to ∞.

Parameters: μ is a location parameter, $\sigma > 0$ is the scale parameter.

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$

Range: $-\infty < t < \infty$

PDF: $$f(t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(t-\mu)^2}{2\sigma^2}\right]$$

SDF: $$S(t) = 1 - \Phi\left(\frac{t-\mu}{\sigma}\right),$$

Mean: μ

Median: μ

Mode: μ

Variance: σ^2

Notes: An accelerated failure time specification of covariates is created for the normal distribution by modeling μ as `FORM = LOGLIN` and specifying a `COVAR` list. A probit model is estimated when for every observation, either t_u=`NEGINFINITY` or t_e = `INFINITY`.

References: Christensen (1984), Evans et al. (1993)

See also: LOGNORMAL, BIVNORMAL

PARETO

Parameters:	b (scale), c (shape).
Time variables:	t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.
Range:	$b \le t < \infty$
PDF:	$f(t) = \dfrac{b^c c}{a^{c+1}}$
SDF:	$S(t) = [t/b]^{-c}$
Quantile:	$t_q = b(1-q)^{-1/c}$
Mean:	$bc/(c-1), \quad c > 1$
Median:	$b\sqrt[c]{2}$
Mode:	b
Variance:	$\dfrac{b^2 c}{(c-2)(c-1)^2}, \quad c > 2$
Constraints:	$b \ge 0$, $c > 0$
References:	Christensen (1984), Evans et al. (1993)
See also:	LOWMAX

PASCAL

This is the Pascal distribution, which is the discrete version of the negative binomial distribution.

Parameters: p (probability), n (count).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: integer $t \geq 0$

PDF: $$f(t) = \binom{t+n-1}{n-1} p^n (1-p)^t$$

SDF: $S(t) = \beta_p(n, t+1)$

Mean: $n(1-p)/p$

Mode: Floor$[p(n-1)/(1-p)]$

Variance: $n(1-p)/p^2$

Constraints: $0 \leq p \leq 1$, integer $n > 0$

Reduced models: Reduces to the geometric distribution when $n = 1$.

References: Christensen (1984), Evans et al. (1993)

See also: NEGBINOMIAL, GEOMETRIC, POWERSERIES, BINOMIAL

POISSON

This is the discrete Poisson distribution.

Parameters:	λ (hazard)
Time variables:	t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.
Range:	integer $t \geq 0$
PDF:	$f(t) = \frac{\lambda^t e^{-\lambda}}{t!}$
SDF:	$S(t) = \frac{\Gamma(t+1,\lambda)}{\Gamma(t+1)}$
Mean:	λ
Mode:	Floor(λ)
Variance:	λ
Constraints:	$\lambda > 0$
References:	Christensen (1984), Evans et al. (1993)
See also:	POWERSERIES

POWERFUNCTION

Parameters: a (location), b (scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $a \le t \le a + b$

PDF: $$f(t) = \frac{c}{b}\left(\frac{t-a}{b}\right)^{c-1}$$

SDF: $$S(t) = 1 - \left(\frac{t-a}{b}\right)^{c}$$

Quantile: $t_q = a + bq^{1/c}$

Mean: $a + bc/(c + 1)$

Median: $a + b2^{-1/c}$

Mode: $$\begin{cases} a+b, & c \ge 1 \\ a+b/2, & c = 1 \\ a, & c < 1 \end{cases}$$

Variance: $$\frac{b^2 c}{(c+2)(c+1)^2}$$

Constraints: $b > 0$, $c \ge 0$

Reduced models: Reduces to the uniform distribution when $c = 1$.

References: Christensen (1984), Evans et al. (1993)

See also: REVPOWERFUNCTION, LOGISTIC, WEIBULL, GUMBEL, BETA, PARETO

RAISEDCOSINE

This is the raised cosine distribution.

Parameters: a (location), b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $-\pi b + a \le t \le \pi b + a$

PDF: $$f(t) = \frac{1 + \cos\left(\frac{t-a}{b}\right)}{2b}$$

SDF: $$S(t) = \frac{1}{2} - \frac{1}{\pi}\left(\frac{t-a}{b}\right) - \frac{1}{\pi}\sin\left(\frac{t-a}{b}\right)$$

Mean: a

Median: a

Mode: a

Variance: $b^2(\pi^2/3 - 2)$

Constraints: $b \ge 0$

References: Christensen (1984)

RANDOMWALK

This is the random walk distribution.

Parameters: b (scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t > 0$

PDF: $$f(t) = \frac{1}{\sqrt{2\pi bt}} \exp\left[-\frac{c^2 t}{2b} \left(\frac{b}{t} - \frac{1}{c} \right)^2 \right]$$

SDF: $$S(t) = 1 - \Phi\left(\frac{bc - t}{\sqrt{bt}} \right) - e^{2c} \Phi\left(\frac{-bc - t}{\sqrt{bt}} \right)$$

Mean: $b\,(1 + c)$

Mode: $b\sqrt{c^2 + 1/4} - b/2$

Variance: $b^2\,(2 + c)$

Constraints: $b > 0$, $c > 0$

References: Christensen (1984), Wise (1966)

See also: INVGAUSSIAN

RAYLEIGH

Parameter: b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t > 0$

PDF: $$f(t) = \frac{t}{b^2} \exp\left[-\frac{t^2}{2b^2} \right]$$

SDF: $$S(t) = \exp\left[-\frac{t^2}{2b^2} \right]$$

Hazard: $h(t) = t/b^2$

Median: $b\sqrt{\log(4)}$

Mode: b

Constraints: $b > 0$

References:

REVPOWERFUNCTION

This is the reversed power distribution.

Parameters: a (location), b (scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $a \le t \le a + b$

PDF: $$f(t) = \frac{c}{b}\left(\frac{a+b-t}{b}\right)^{c-1}$$

SDF: $$S(t) = \left(1 - \frac{t-a}{b}\right)^{c}$$

Quantile: $t_q = a + b[1-(1-q)^{1/c}]$

Mean: $a + b/(c + 1)$

Median: $$a + b\left(1 - 2^{-1/c}\right)$$

Mode: $$\begin{cases} a, & c > 1 \\ a + b/2, & c = 1 \\ a + b, & c < 1 \end{cases}$$

Variance: $$\frac{b^2 c}{(c+2)(c+1)^2}$$

Constraints: $b > 0$, $c \ge 0$

Reduced models: Reduces to the uniform distribution when $c = 1$.

References: Christensen (1984).

See also: POWERFUNCTION

RINGINGEXP0

This is the ringing exponential distribution at phase 0 degrees.

Parameters: a (location), b (scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq a$

PDF: $$f(t) = \frac{1+2c}{b(1+c)} e^{-\frac{t-a}{b}} \cos^2\left(\frac{t-a}{b}\sqrt{\frac{c}{2}}\right)$$

SDF: $$S(t) = \frac{e^{-\frac{t-a}{b}}}{1+c}\left[\cos^2\left(\frac{t-a}{b}\sqrt{\frac{c}{2}}\right) - \sqrt{\frac{c}{2}}\sin\left(\frac{t-a}{b}\sqrt{2c}\right)\right]$$

Mean: $$a + \frac{b\left(1+c+2c^2\right)}{1+3c+2c^2}$$

Mode: a

Variance: $$\frac{b^2\left(1+7c^2+16c^3+4c^4\right)}{\left(1+3c+2c^2\right)^2}$$

Constraints: $b > 0$, $c \geq 0$

References: Christensen (1984)

See also: RINGINGEXP180

RINGINGEXP180

This is the ringing exponential distribution at phase 180 degrees.

Parameters: a (location), b (scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq a$

PDF: $$f(t) = \frac{2c+1}{bc} e^{-\frac{t-a}{b}} \sin^2\left(\frac{t-a}{b}\sqrt{\frac{c}{2}}\right)$$

SDF: $$S(t) = \frac{e^{-\frac{t-a}{b}}}{c}\left[c + \sin^2\left(\frac{t-a}{b}\sqrt{\frac{c}{2}}\right) + \sqrt{\frac{c}{2}}\sin\left(\frac{t-a}{b}\sqrt{2c}\right)\right]$$

Mean: $$a + \frac{b(3+2c)}{1+2c}$$

Mode: $$a + \frac{2b\arctan\left(\sqrt{2c}\right)}{\sqrt{2c}}$$

Variance: $$\frac{b^2\left(3+4c^2\right)}{\left(1+2c\right)^2}$$

Constraints: $b > 0$, $c > 0$

References: Christensen (1984)

See also: RINGINGEXP0

SHIFTEXPONENTIAL

The exponential is commonly used in reliability engineering, queuing theory and biology. The 'memoryless' property of the exponential distribution is an important characteristic. It says, in effect, that for a survivor, future times to failure are completely independent of the past. This form of the exponential distribution provides for a location parameter. Also, this version uses a scale parameter b, rather then a hazard parameter λ ($b = 1/\lambda$).

Parameter:	a (location), b (scale = $1/\lambda$).
Time variables:	t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$
Range:	$t \geq a$
PDF:	$f(t) = \exp(-(t - a)/b)/b$
SDF:	$S(t) = \exp(-(t - a)/b)$
Hazard:	$h(t) = 1/b$
Quantile:	$t_q = a - b\ln(1 - q)$
Mean:	$a + b$
Median:	$a + b\ln(2)$
Mode:	a
Variance:	b^2
Constraints:	$b > 0$
References:	Christensen (1984), Evans et al. (1993), Nelson (1982)
See also:	The EXPONENTIAL distribution is a 1 parameter (hazard) version of this distribution.

SHIFTGAMMA

This is the three parameter (shifted) gamma distribution, also known as the Pearson Type III distribution.

Parameters: a (location), b (scale, inverse of the hazard), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq a$

PDF: $$f(t) = \frac{\left(\frac{t-a}{b}\right)^{c-1} e^{-\frac{t-a}{b}}}{b\Gamma(c)}$$

SDF: $$S(t) = \frac{\Gamma\left(c, \frac{t-a}{b}\right)}{\Gamma(c)}$$

Hazard: $$h(t) = \frac{\left(\frac{t-a}{b}\right)^{c-1} e^{-\frac{t-a}{b}}}{b\Gamma\left(c, \frac{t-a}{b}\right)}$$

Quantile: $t_q = a + \frac{b}{2}\chi^2_q(2c)$

Mean: $a + bc$

Mode: $a + b(c-1)$, $c > 1$

a, $c \leq 1$

Variance: b^2c

Constraints: $b \geq 0$, $c \geq 0$

Reduced models: Reduces to a shifted exponential distribution when $c = 1$. Reduces to an Erlang distribution with integer parameter c. Reduces to a Chi-squared distribution with v degrees of freedom with $a = 2$, $b = v/2$. Reduces to a gamma distribution with $a = 0$ and $b = 1/\lambda$.

References: Christensen (1984), Elandt-Johnson and Johnson (1980), Evans et al. (1993), Kalbfleisch and Prentice (1980).

See also: SHIFTEXPONENTIAL, GAMMA, GENGAMMA, CHISQUARED

SHIFTLOGNORMAL

This is a three-parameter shifted lognormal distribution, which is a reparameterization of the LOGNORMAL distribution.

Parameters: a (location), b (scale), c (shape).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$

Range: $t \geq a$

PDF: $$f(t) = \frac{1}{(t-a)c\sqrt{2\pi}} e^{\left[-\frac{1}{2c^2}\ln\left(\frac{t-a}{b}\right)^2\right]}$$

SDF: $$S(t) = 1 - \Phi\left[\frac{1}{c}\ln\left(\frac{t-a}{b}\right)\right]$$

Mean: $a + b\exp(c^2/2)$

Median: $a + b$

Mode: $a + b\exp(c^2/2)$

Variance: $b^2\exp(c^2)[\exp(c^2 + 2)]$

Constraints: $b > 0$, $c > 0$

Notes: This distribution is related to the SHIFTLOGNORMAL distribution as follows. The two-parameter LNNORMAL cumulative density is found from the Normal density by taking $\Phi[\ln(t - a)/b]$ whereas in the three parameter SHIFTLOGNORMAL we take $\Phi\{\ln[(t - a)/b]/c\}$.

References: Nelson (1982)

See also: NORMAL, LOGNORMAL

SHIFTWEIBULL

This is the three-parameter Shifted Weibull distribution.

Parameters: a (location) b (scale and characteristic life= 63rd percentile), c (shape).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq a$

PDF: $$f(t) = \frac{c}{b}\left(\frac{t-a}{b}\right)^{c-1} \exp\left[-\left(\frac{t-a}{b}\right)^{c}\right]$$

SDF: $$S(t) = \exp\left[-\left(\frac{t-a}{b}\right)^{c}\right]$$

Hazard: $h(t) = ct^{c-1}b^{-c}$

Mean: $a + b\Gamma(1 + 1/c)$

Median: $a + b\sqrt{\ln(2)}$

Mode: $$\begin{cases} a + b\sqrt[c]{1-1/c}, & c > 1 \\ a, & c \leq 1 \end{cases}$$

Variance: $$b^2\left\{\Gamma(1+2/c) - \left[\Gamma(1+2/c)\right]^2\right\}$$

Constraints: $b > 0$, $c > 0$

Reduced models: Reduces to the exponential distribution when $c = 1$, reduces to the Rayleigh distribution when $c = 2$.

References: Christensen (1983), Nelson (1982)

See also: WEIBULL

SILER

This is the Siler distribution frequently used as a competing hazards model for mortality (see Gage 1989).

Parameters: a_1, b_1 ("infant mortality" component), a_2 (constant), a_3, b_3 (senescent mortality component).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = \left(a_1 e^{-b_1 t} + a_2 + a_3 e^{b_3 t}\right) \exp\left[-\frac{a_1}{b_1}(1 - e^{-b_1 t}) - a_2 t + \frac{a_3}{b_3}(1 - e^{b_3 t})\right],$$

SDF: $$S(t) = \exp\left[-\frac{a_1}{b_1}(1 - e^{-b_1 t}) - a_2 t + \frac{a_3}{b_3}(1 - e^{b_3 t})\right]$$

Hazard: $h(t) = a_1 \exp(-b_1 t) + a_2 + a_3 \exp(b_3 t)$

Constraints: $a_1 \geq 0$, $a_2 \geq 0$, $a_3 \geq 0$, $b_1 \geq 0$, $b_3 \geq 0$

Reduced models: $a_1 = 0$ reduces to a Gompertz-Makeham. $a_1 = 0$, $a_2 = 0$ reduces to a Gompertz. Reduces to an exponential in a number of ways.

References:

See also: EXPONENTIAL, GOMPERTZ, MAKEHAM, MIXMAKEHAM

SMALLEXTREME

Parameters: a (location) and b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $-\infty < t < \infty$

PDF: $$f(t) = b^{-1} \exp\left[\frac{t-a}{b} - \exp\left(\frac{t-a}{b}\right)\right]$$

SDF: $$S(t) = \exp\left\{-\exp\left(\frac{t-a}{b}\right)\right\}$$

Hazard: $$h(t) = b^{-1} \exp\left[\frac{t-a}{b}\right]$$

Mean: $a - kb$ k=0.57721... is Euler's constant

Median: $a - b \log[\log(2)]$

Mode: a (63.2nd percentile)

Variance: $b^2\pi^2/6$

Constraints: $b \geq 0$

References: Evans et al. (1993), Nelson (1982)

STERILE or IMMUNE

This distribution is used to form degenerate distributions of the forms $f(t) = (1 - p)f_1(t)$ and $S(t) = (1\text{-}p)S_1(t) + p$. This is done by using the `MIX` function with STERILE and specifying a mixture of $f_1(t)$ and the STERILE distribution. Given failure times t_u and t_e and, perhaps, the left truncation limits t_α and t_ω, this distribution returns:

0 if an exact failure or an interval censored failure occurs: $t_e < t_\omega$

1 if a right censored observation occurs: $t_e \geq t_\omega$

In words, the distribution returns 0 if there is no possibility that the observation may have been "sterile" (because some failure was observed), and returns 1 if there is a possibility that the observation was a "sterile" observation (right censored observation). This distribution has no intrinsic parameters, and covariates cannot be modeled on the hazard function of this distribution.

Example. Suppose individuals fail with an underlying normal distribution, but the population is contaminated by an unknown fraction, p, of non-susceptible individuals. The ***mle*** code is:

```
MODEL
  MIX
    PARAM  p  LOW=0  HIGH=1  FORM=NUMBER  END,
    PDF NORMAL( last_alive  first_dead )
      PARAM mu      LOW=100  HIGH=200  START=150  END
      PARAM sigma   LOW=0.1  HIGH=20   START=10   END
    END {pdf}
    ,
    PDF STERILE( last_alive  first_dead ) END
END
```

Call L_n the likelihood from the normal PDF. Then, the likelihood for an exact failure or interval censored failure will be pL_n. For right censored observations, the likelihood will be $pL_n + (1–p)$.

References: Holman (1995, 1998), Nelson (1982)

SUBBOTIN

This is the Subbotin distribution, which includes a number of important distributions as special cases, including the uniform, Laplace, and normal distributions. The distribution becomes discontinuous at the median when $c < 1$.

Parameters: a (location), b (scale), c (shape)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $-\infty < t < \infty$

PDF: $$f(t) = \frac{1}{b2^{1+1/c}\Gamma(1+1/c)} e^{-\frac{1}{2}\left|\frac{t-a}{b}\right|^c}$$

SDF: $$S(t) = \frac{1}{2} - \frac{\operatorname{sgn}(t-a)\gamma\left(\frac{1}{c}, \frac{1}{2}\left|\frac{t-a}{b}\right|^c\right)}{\Gamma(1/c)}$$

Mean: a

Median: a

Mode: a

Variance: $$\frac{b^2 2^{2/c}\Gamma(3/c)}{\Gamma(1/c)}$$

Constraints: $b \geq 0$, $c > 0$

Reduced models: Approaches the rectangular distribution as $c \to \infty$, reduces to the Laplace distribution when c is 1, and reduces to the normal distribution when c is 2.

References: Christensen (1984), Subbotin (1923)

See also: NORMAL, LAPLACE, UNIFORM

UNIFORM or RECTANGULAR (Continuous)

The continuous uniform distribution has no parameters. However, truncation limits t_α and t_ω should be specified. When truncation limits are not specified (i.e. only one or two time variables are specified), then the standard uniform distribution is assumed and used: $t_\alpha = 0$ and $t_\omega = 1$.

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t_\alpha \le t < t_\omega$

PDF: $f(t) = 1/(t_\omega - t_\alpha)$

SDF: $$S(t) = \frac{t_\omega - t}{t_\omega - t_\alpha}$$

Hazard: $h(t) = 1/(t_\omega - t)$

Mean: $(t_\omega - t_\alpha) / 2$

Median: $(t_\omega - t_\alpha) / 2$

Variance: $(t_\omega - t_\alpha)^2 / 12$

References: Evans et al. (1993), Nelson (1982)

VONMISES

This is the von Mises distribution, which is the analog of a normal distribution on a circular range.

Parameters: a (location), b (scale)

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $0 < t < 2\pi$

PDF: $$f(t) = \frac{\exp\left[b\cos(t-a)\right]}{2\pi \mathrm{I}_0(b)}$$

SDF: $$S(t) = 1 - \frac{t\mathrm{I}_0(b) + 2\sum_{j=1}^{\infty}\left[\mathrm{I}_j(b)\left[\sin\left(jt - ja\right)\right]j^{-1}\right]}{2\pi \mathrm{I}_0(b)}$$

Mean direction: a

Median: a

Mode: a

Antimode: a

Circular variance: $1 - \mathrm{I}_1(b)/\mathrm{I}_1(b)$

Constraints: $0 < a < 2\pi$, $b > 0$

References: Evans et al. (1993), Rao (1973)

WEIBULL

This is the Weibull distribution, sometimes called the generalized Rayleigh distribution.

Parameters: b (scale and characteristic life= 63rd percentile), c (shape).

Time variables: t_u, t_e, t_α, t_ω. An exact failure is defined when $t_u = t_e$.

Range: $t \geq 0$

PDF: $$f(t) = ct^{c-1}b^{-c}\exp\left[-\left(\frac{t}{b}\right)^c\right]$$

SDF: $$S(t) = \exp\left[-\left(\frac{t}{b}\right)^c\right]$$

Hazard: $h(t) = ct^{c-1}b^{-c}$

Mean: $b\Gamma(1 + 1/c)$

Median: $b\sqrt{\ln(2)}$

Mode: $$\begin{cases} b\sqrt[c]{1-1/c}, & c > 1 \\ 0, & c \leq 1 \end{cases}$$

Variance: $$b^2\left\{\Gamma(1+2/c) - \left[\Gamma(1+2/c)\right]^2\right\}$$

Constraints: $b > 0$, $c > 0$

Reduced models: Reduces to the exponential distribution when $c = 1$, reduces to the Rayleigh distribution when $c = 2$.

References: Christensen (1983), Evans et al. (1993), Nelson (1982)

See also: SHIFTWEIBULL

NUMBERS, SYMBOLS, CONSTANTS, FUNCTIONS, AND CONVERSIONS

Symbols

t	Used to denote the random variable, whether continuous or discrete. Typically, this is a time variable.
ρ	A parameter that defines a correlation coefficient.
σ	A scale parameter that defines the standard deviation of the distribution.
μ	A location parameter that also defines the mean of the distribution.
a	A location parameter
b	A scale parameter
c	A shape parameter
h	A parameter that is directly interpretable as a hazard.
p	A parameter that is directly interpretable as a probability.
t_q	The qth quantile: $[1–S(t_q)] = q$.
$f(t)$	The probability density function (PDF).
$S(t)$	The survival distribution (SDF): $S(t) = 1 - \int_0^t f(x)dx = \int_t^\infty f(x)dx$.
$h(t)$	The hazard function: $h(t) = \frac{f(x)}{S(t)}$.
$\frac{d[g(x)]}{dx}$	The first derivative of the function $g(x)$

Constants

π	PI, The value pi ≈ 3.141 592 653 589 793 238 462 643
e	E, The base of the natural log ≈ 2.718 281 828 459 045 235 360 287
γ	EULERSC, Euler's constant ≈ 0.577 215 664 901 532 860 606 512

Function definitions

$\chi_q^2(x)$ The Chi-squared q quantile: $\Pr\left[\chi^2 \le \chi_q^2(x)\right] = q$

Φ_q The standard normal q quantile: $\Pr(x \le \Phi_q) = q$

$\Phi(x)$ The standard normal cumulative density function: $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{u^2}{2}} du$.

erf(x) The error function, ERF(x): $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du = 2\Phi\left(x\sqrt{2}\right) - 1$.

$I_k(x)$ The modified Bessel function of the 1st kind, order k, BESSELI(k, x):

$$I_k(x) = \frac{x^k}{2^k} \sum_{j=0}^{\infty} \frac{\left(\frac{1}{4}x^2\right)^j}{j!\Gamma(1+j+k)}.$$

$K_k(x)$ The modified Bessel function of the 2nd kind, order k, BESSELK(k, x):

$$K_k(x) = \pi \frac{I_{-k}(x) - I_k(x)}{2\sin(i\pi)}.$$

B(ν, ω) The beta function BETAF(ν, ω): $B(\nu, \omega) = \int_0^1 z^{\nu-1}(1-x)^{\omega-1} dx = \frac{\Gamma(\nu)\Gamma(\omega)}{\Gamma(\nu+\omega)}$.

B_p(ν, ω) The normalized incomplete beta function, IBETA(ν, ω):

$$B_p(\nu, \omega) = \frac{\int_0^p x^{\nu-1}(1-x)^{\omega-1} dx}{B(\nu, \omega)}$$

β_p(ν, ω) The complement of the normalized beta function (IBETAC): $\beta_p(\nu, \omega) = 1 - B_p(\nu, \omega)$.

Γ(ν) The gamma function (GAMMAF): $\Gamma(\nu) = \int_0^\infty x^{\nu-1} e^{-x} dx$.

γ(ν, ω) The incomplete gamma function of the 1st kind: $\gamma(\nu, \omega) = \int_0^\omega x^{\nu-1} e^{-x} dx$.

The IGAMMA(x, y) function returns γ(ν, ω)/Γ(ν)

Γ(ν, ω) The incomplete gamma function of the 2nd kind: $\Gamma(\nu, \omega) = \int_\omega^\infty x^{\nu-1} e^{-x} dx$.

The IGAMMAC(x, y) function returns Γ(ν, ω)/Γ(ν) = 1 – γ(ν, ω)/Γ(ν)

ψ(x) The digamma function. $\psi(x) = \frac{d\left[\ln\Gamma(x)\right]}{dx}$.

ψ′(x) The trigamma function. $\psi'(x) = \frac{d\left[\psi(x)\right]}{dx} = \frac{d^2\left[\ln\Gamma(x)\right]}{dx^2}$.

$\ln(x)$	The natural (Napierian) log of x. LN(x) and LOG(x)
$\delta(x, y)$	Kronecker's delta function: $\delta(x,y)=\begin{cases}1, & x=y\\ 0, & x\neq y\end{cases}$. DELTA($x$, y).
$\binom{n}{k}$	Combinations of n taken k at a time = $\binom{n}{k}=\frac{n!}{(n-r)!}$. COMB($n$, k)

The Greek alphabet

A	α	alpha	I	ι	iota	P	ρ	rho
B	β	beta	K	κ	kappa	Σ	σ	sigma
Γ	γ	gamma	Λ	λ	lambda	T	τ	tau
Δ	δ	delta	M	μ	mu	Y	υ	upsilon
E	ε	epsilon	N	ν	nu	Φ	ϕ	phi
Z	ζ	zeta	Ξ	ξ	xi	X	χ	chi
H	η	eta	O	o	omikron	Ψ	ψ	psi
Θ	θ	theta	Π	π	pi	Ω	ω	omega

Metric prefixes

10	deka (da)	10^{-1}	deci (d)
10^{2}	hecto (h)	10^{-2}	centi (c)
10^{3}	kilo (k)	10^{-3}	milli (m)
10^{6}	mega (M)	10^{-6}	micro (μ)
10^{9}	giga (G)	10^{-9}	nano (n)
10^{12}	tera (T)	10^{-12}	pico (p)
10^{15}	peta (P)	10^{-15}	femto (f)
10^{18}	exa (E)	10^{-18}	atto (a)

Temperature conversions

	a °C	*b* °K	*c* °F
a °C	*a*	$a = b - 273.15$	$a = (c - 32)/1.8$
b °K	$b = a + 273.15$	*b*	$b = (c + 459.67)/1.8$
c °F	$c = 1.8a + 32$	$c = 1.8b - 459.67$	*c*

Selected Systéme International d'Unités

Quantity	SI Unit	Derivation	Other units
acceleration	m/s^2		
angular acceleration	rad/s^2		
angular velocity	rad/s		
capacitance (electrical)	farad, F	A×s/V	
charge (electrical)	coulomb, C	A×s	electrostatic units, esu = $3^{-1}\times10^{-9}$ C
current (electrical)	ampre, A		
density	kg/m^3		
energy, work, heat	joule, J	N×m	calorie, cal = 4.184 J British thermal unit, BTU = 1055.87 J foot-pound, ft-lb = 1.35582 J electronvolt, eV = 1.60219×10^{-19} J erg = 10^{-7} J
field strength (electrical)	V/m		
flux of light	lumen, lm	cd×sr	
force	newton, N	$kg\times m/s^2$	dyne, dyn = 10^{-5} N
frequency	hertz, Hz	1/s	cycles per second, cps = 1 Hz
illumination	lux, lx	lm/m^2	
inductance	herny, H	V×s/A	
length	meter, m		angstrom (A) = 10^{-10} M
luminance	candela/m^2, cd/m^2		
magnetic field strength	A/m		oersted, Oe = $4^{-1}\times10^{3}$ A/m
magnetic flux	weber, Wb	V/s	maxwell, Mx = 10^{-8} Wb
magnetic flux density	tesla, T	Wb/m^2	gauss, G = 10^{-4} T
magnetomotive force	ampre, A		
mass	kilogram, kg		
power	watt, W	J/s	horsepower, hp = 745.7 W
pressure	N/m^2		atmosphere, atm = 1.01325×10^{5} N/m^2 bar = 10^5 N/m^2 kilopascals, kPa = 1000 N/m^2
velocity	m/s		
voltage, electromotive force, electrical potential	volt, V	W/A	

Angles

	"	'	°	radians	grads
"	1	0.0166667	0.000277778	0.0159155	0.0176839
'	60	1	0.0166667	0.954930	1.06103
°	3600	60	1	57.2958	63.6620
radians	62.8319	1.04720	0.0174533	1	1.11111
grads	56.5487	0.942478	0.0157080	0.9	1

Time

	second	minute	hour	day	week	year
second	1	0.0166667	0.000277778	1.15741E-05	1.65344E-06	3.17969E-08
minute	60	1	0.0166667	0.000694444	9.92063E-05	1.90781E-06
hour	3600	60	1	0.0416667	0.00595238	0.000114469
day	86400	1440	24	1	0.142857	0.00274725
week	604800	10080	168	7	1	0.0192308
year	31449600	524160	8736	364	52	1

Avoirdupois weight

	kg	lb	ounce	dram	grain	short ton	long ton
kg	1	2.20462	35.2740	564.383	15432.4	0.00110231	0.000984207
lb	0.45359237	1	16	256	7000	0.000500000	0.000446429
ounce	0.0283495	0.0625000	1	16	437.500	0.0000312500	2.79018E-05
dram	0.00177185	0.00390625	0.0625	1	27.3438	1.95313E-06	1.74386E-06
grain	6.47989E-05	0.000142857	0.00228571	0.0365714	1	7.14286E-08	6.37755E-08
short ton	907.185	2000	32000	512000	14000000	1	0.892857
long ton	1016.05	2240	35840	573440	15680000	1.12	1

Long measure

	meter	inch	foot	yard	rod	fulong	mile	nautical mile	leagues
meter	1	39.370	3.2808	1.0936	6.0149	240.59	1924.8	2221.6	5774.3
inch	0.02540	1	0.083333	0.027778	0.15278	6.111111	48.889	56.428	146.67
foot	0.30480	12	1	0.33333	1.8333	73.33333	586.67	677.14	1760
yard	0.91440	36	3	1	5.5	220	1760.0	2031.4	5280
rod	0.16625	6.5455	0.54545	0.18182	1	40	320	369.35	960
fulong	0.0041564	0.16364	0.013636	0.0045455	0.025	1	8	9.2337	24
mile	0.0005195 5	0.020455	0.0017045	0.00056818	0.0031250	0.125	1	1.1542	3
nautical mile	0.0004501 3	0.017722	0.0014768	0.00049227	0.0027075	0.1082992	0.86639	1	2.5992
leagues	0.0001731 8	0.0068182	0.00056818	0.00018939	0.0010417	0.0416667	0.33333	0.38474	1

fluid flow (volume/time)

	m^3/s	Mgal/day	ft^3/sec	gal/min	acre-ft/day
m^3/s	1	22.8	35.3	15850	70.0
Mgal/day	0.0438	1	1.55	694	3.07
ft^3/sec	0.0283	0.656	1	448	1.98
gal/min	6.31×10^{-5}	0.00144	0.00223	1	0.00442

acre-ft/day	0.0143	0.326	0.504	226	1

Power (energy/time)

	joule/sec	ft-lb_{force}/sec	kWatt	horsepower	Btu/sec
joule/sec	1	0.738	0.001	0.00134	9.48×10^{-4}
ft-lb_{force}/sec	1.36	1	0.00136	0.00182	0.00128
kWatt	1000	738	1	1.34	0.948
horsepower	746	550	0.746	1	0.707
Btu/sec	1055	778	1.05	1.41	1

Kinematic Viscosity

	m^2/s	cm^2/sec	ft^2/sec	cenitstoke
m^2/s	1	10^4	10.7	10^6
cm^2/sec	10^{-4}	1	0.00107	100
ft^2/sec	0.0929	929	1	9.34×10^4
cenitstoke	10^{-6}	0.01	1.07×10^{-5}	1

ERROR AND WARNING MESSAGES

A number of error and warning messages are produced by ***mle***. Warning messages are given for models, parameters and other iterative functions that might not completely converge. Error messages cause ***mle*** to stop running. They can be roughly divided into those that come from the run-time routines, the symbol table routines, the parsing routines, and the mathematical library. Finally, ***mle*** has a number of help messages and other messages that occur in response to improper command line options.

Messages from command line options

The following message is printed when command line options are not recognized and at least one of the options is taken as an input file. For example, typing `mle xxx` yields:

```
Error: File "xxx" does not exist

Usage: mle [-v] [-p] [-i] [-dd] [-ds] [-dp] [-di] [-dl] [-d #] [mlefile]
  -v sets verbose on. Iteration histories are printed
  -p only parses the mle file
  -i runs mle interactively
  -dd turns on data debugging
  -ds turns on symbol table debugging
  -dp turns on parser debugging
  -di turns on integration debugging
  -dl turns on likelihood debugging
  -d sets debugging to level #
  mlefile is the name of the file with the program

Usage: mle -h [name1 name2 . . . .]
  help for PDFs, functions, symbols, parameter transforms
  -h matches words exactly, -H searches within words

Usage: mle -pn n1 n2 . . . .
  parses n's and returns values and type
```

File <name> does not exist. Try again.

Typing `mle` on the command line will result in the message

```
mle Program file to run?
```

Should you type a file name that does not exist, the following message appears:

```
  File asd does not exist.  Try again.
mle Program file to run?
```

Ensure the proper directory and file extension is being used. Note that ***mle*** does not automatically append `.mle` to the input file.

Warning messages

Warning messages come from routines that iteratively attempt to find a solution of parameters or other functions. Warning messages will not result in termination of the ***mle*** run.

Warning FINDMIN reached maximum iterations

Warning FINDZERO reached maximum iterations

The FINDMIN or FINDZERO function was not evaluated to the specified tolerance in the specified number of iterations. You can increase the number of iterations to the function (one way is by increasing the value of FIND_MAXITS) or decrease the convergence criterion (one way is by decreasing the value of FIND_EPS).

Warning: gamma SDF (by continued fractions) did not completely converge

Warning: gamma SDF (by series) did not completely converge

These two messages suggest that some evaluation of the Euler's incomplete gamma function was not very precise.

Warning: beta CDF did not converge

This message suggests that some evaluation of the incomplete Beta function was not very precise.

Warning: Upper [Lower] Interval for param x did not converge to x.xxx in yyyy iterations

This message arises when ***mle*** has troubles finding the upper or lower limit of a likelihood confidence interval. The number of iterations should be increased (set CI_MAXITS to a higher value), or the convergence criterion should be relaxed (set CI_CONVERGE to a larger value).

Warning: Upper [Lower] Interval for param x not bound between xxx and yyy.

This message arises when a confidence interval is larger then the upper and lower limits of the parameter. The `HIGH` or `LOW` limits for the parameter should be changed so that the confidence limit is within the limits of the parameters.

Warning: the matrix is singular

This message indicates that the variance-covariance matrix could not be computed because the observed Fisher's information matrix was singular. This occurs when one or more parameters have very large standard errors, or changes in the parameter do not affect the likelihood. Some suggestions are: reduce the number of parameters, ensure all parameters affect the likelihood, solve the likelihood to a higher precision, transform one or more parameters so that the parameter estimate is not near a mathematical limit. This last situation occurs, for example, when a probability is modeled untransformed (between 0 and 1) and the parameter estimate is near 0 or 1. Using a logistic specification for the parameter will sometimes fix the problem.

Run-time errors

Error (run time): Tried assigning null string to char var <name>

A null string (i.e. a string of length zero specified by "") cannot be assigned to a character variable.

Error (run time): Unimplemented or unknown METHOD: <name>)

The requested maximization method (set by METHOD=<name>) is not recognized.

Error (run time): Unimplimented integration method: <name>

The requested integration method (set by INTEGRATE_METHOD=<name>) is not recognized.

Error (run time): Bad string in STRING2REAL

A string argument to the STRING2REAL function could not be converted into a real number.

Error (run time): Bad string in STRING2INT

A string argument to the STRING2INT function could not be converted into an integer.

Error (run time): Bad real ident. <name> is type <type>

Error (run time): Bad integer identifier type found

An argument to an integer function was not an integer.

Error (run time): Bad string identifier type found

An argument to a string function was not a string or character.

Error (run time): Calling boolean func <name> with [real, integer, string/char, boolean] args

Error (run time): Type mismatch for arg n calling func <name>

The type (integer, real, boolean, string, character) for argument *n* of function <*name*> was incorrect.

Error (run time): Opening [INFILE, DATAFILE, MLERC] file xxxx: <message>

An error occurred while opening a file. The <message> can be one of the following:

File was not found

Path was not found

Too many open files

File access denied

Invalid file reference

Not enough memory

Invalid environment

Invalid drive letter

Can't remove current directory

Can't rename files across drives

Disk read error

Disk write error

Disk is write-protected

Unknown device

Disk drive is not ready

Disk seek or sector error

Unknown media

The printer is out of paper

Error trying to write to the output device

Error trying to read an input device

A hardware failure occurred

Errors from the parser

Error messages from the parser always contain information on the line and column where the error occurred.

Error found while parsing <id> at line <line#> column <column#>

Expected a positive constant instead of "<text>"

A positive constant is expected in the FIELD and LINE specifications of the DATA statement.

Boolean expression was expected

A boolean expression was expected but not found. For example, as the first expression in the IF...THEN...ELSE...END function must be a boolean expression.

"<name>" must be previously declared for use here

A variable used in an expression had not been previously declared. All variables must be declared, either as a predeclared variable, in a PARAM function, in the DATA statement, or in an ASSIGN statement before being used in an expression.

"<name>" exists and cannot be declared as a PARAM

An attempt was made to declare <name> as a parameter, but it was previously declared.

<name> doesn't exists, so it can't be reduced.

The parameter <name> found in a REDUCE statement does not exist.

Bad argument type to [function] <func>. Expected <type> but found <type>

An argument to the named function is not correct.

<variable> already exists. It cannot be a DATA variable.

A variable defined in the DATA statement already exists.

Bad type in assign statement

The resulting type on the right-hand side of the assignment is incompatible with the left-hand side.

Can't assign value of type <type> to variable of type <type>.

The two types are incompatible.

Bad number format found while scanning "<text>"

The text was supposed to be converted into a number, but could not be properly converted. Usually there is an invalid character.

Character constant is too long

A character constant had more than one character.

Unclosed comment at end of file

This results when a comment is not properly closed.

Error: bad syntax at: line <line#> column <column#>

The bad token is:<id>

One of the following was expected:

This error occurs when improper syntax is found.

Error messages from data routines

Error (data): Data transformation, bad function type

The function defined for a data transformation does not return a real or integer type.

Error (data): Unexpected end of file <name> reading observation <n> line <n> of <n>, field <n> of <n>

The file ended when an observation was only partially read.

Error (data): Unexpected end of line <name> reading observation <n> line <n> of <n>, field <n> of <n>

The line ended when an observation was only partially read.

Bad value <n1> line <n2> field <n3>. Can't convert: <text> to a number.

The value for observation *n1*, on line *n2* in field *n3* couldn't be properly converted to a number.

Error (data): No data file assigned. Use DATAFILE procedure'

The data file was not assigned using DATAFILE procedure. Place the statement `DATAFILE("<name>")` before the `DATA` statement.

Error messages from function calls:

A number of errors arise from improper values being passed to function calls. Frequently the remedies are to place proper constraints on parameter values, clean the input data, transform data, or add a small positive number to all failure times. The following messages arise from these difficulties

Error (math): Attempted division by zero

Error (math): Attempted to take the square root of a negative number: <-nnnn>

Error (math): Bad BIVNORMAL param rho. Expected:-1<=r<=1 but = <nnn>

Error (math): Bad <name> should be > 0 and < 1 but = <nnn>

Error (math): Bad <name> should be >=1 but = <nnn>

Error (math): Bad <name> should be >= 0 but = <nnn>

Error (math): Bad <name> should be > 0 but = <nnn>

Error (math): Bad <name> cannot be =0 but is

Error (math): Bad <name> should be > -1 and < 1 but = <nnn>

Error (math): Bad <name> should be >= -1 and <= 1 but = <nnn>

Error (math): Attempted log of negative number: <nnn>

Error (math): Bad Logit(<nnn>)

Error (math): Bad arg to: POWER(<nnn>, <nnn>)

Error (math): Integer overflow in FACT

Error (math): Integer overflow in COMBINATION

Error (math): Integer overflow in PERMUTE

Error (math): IBETA: arg is not 0 <= t <= 1, = <nnn>

Error (math): Bad random seed: <nnn>

Error messages from symbol table routines

Error (sym table): Wrong type: can't assign <name> (<type>) to <name> (<type>).

An attempt was made to assign incompatible variables.

Error (sym table): Variable of type <type> is too large

There was not enough memory to allocate a variable.

REFERENCES

Agresti A (1990) *Categorical Data Analysis.* New York: John Wiley and Sons.

Ahuja JC and Nash SW (1967) *Sankhya A* **29**:141-56.

Birnbaum ZW and Saunders SC (1969) *Journal of Applied Probability* **6**:319-27.

Borel E (1925) *Principes et formules classiques du Calcul des Probabilitiés*. :Paris.

Box GEP, Hunter WG, Hunter JS (1978) *Statistics for Experimenters*. New York: John Wiley & Sons.

Bratley P, Fox BL, Schrage LE (1983) *A Guide to Simulation* New York: Springer-Verlag.

Brent RP (1973) Algorithms for minimization without derivatives. *Englewood Cliffs, NJ*: Prentice-Hall.

Cox DR, Oakes D (1984) *Analysis of survival data.* London: Chapman and Hall.

Christensen R (1984) *Data Distributions*. Lincoln, MA: Entropy Ltd.

Daniels HE (1945) *Proc Royal Soc London, Series A* **183**:405-35.

Edwards AWF (1972) *Likelihood*. Cambridge: Cambridge University Press.

Efron B (1982) *The Jackknife, the Bootstrap and Other Resampling Plans.* Philadelphia: Society for Industrial and Applied Mathematics.

Eggenberger F, and Pólya G (1923) *Zeit. fur Angew. Math und Mech.* **1**:179-289.

Elandt-Johnson RC, Johnson NL (1980) *Survival Models and Data Analysis*. New York: John Wiley and Sons.

Fisher RA, Corbet AS, Williams CB (1943) *J Animal Ecology* **12**:42-57.

Fisher RA (1921) On the 'probable error' of a coefficient of correlation deduced from a small sample. *Metron* **1**:3-32.

Forsythe G, Malcolm MA, Moler CB (1977) *Computer Methods for Mathematical Computations.* Englewood Cliffs, NJ: Prentice-Hall.

Gage TB (1989) Bio-methematical approaches to the study of human variation in mortality. *Yrbk Phys Anthropol* **32**:185-214.

Geoffe WL, Ferrier GD, Rogers J (1994) Global optimization of statistical functions with simulated annealing. *Journal of Econometrics* **60**:65-99.

Gompertz B (1825) *Phil Trans Roy Soc* London, **115**:513-85.

Gumbel EJ (1947) *Annals Mathematical Statistics* **18**:384-412.

Hammes LM, Treloar AE (1970) Gestational interval from vital records. *Am J Pub Health* **60**:1496-505.

Hazelrig JB, Turner ME, Blackstone EH (1982) Parametric survival analysis combining longitudinal and cross-sectional censored and interval-censored data with concomitant information. *Biometrics* **38**:1-15.

Hilborn R and Mangel M (1997) *The Ecological Detective Confronting Models with Data*. Monographs in Population Biology 28. Princeton, N.J.: Princeton University Press.

Holman DJ (1996) *Total Fecundability and Fetal Loss in Rural Bangladesh*. Doctoral Dissertation, The Pennsylvania State University.

Holman DJ and Jones RE (1998) Longitudinal analysis of deciduous tooth emergence II: Parametric survival analysis in Bangladeshi, Guatemalan, Japanese and Javanese children. American Journal of Physical Anthropology 105(2):209-30.

Kalbfleisch JD, Prentice RL (1980) *The Statistical Analysis of Failure Time Data*. New York: John Wiley & Sons.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. *Science* **220** (4598):671-80.

References

Kishor ST (1982) Probability and Statistics with Reliability, Queuing, and Computer Science Applications. Englewood Cliffs, NJ: Prentice-Hall.

Laplace PS (1774) *Mém. de Math et Phys., l'Acad. Roy. des Sci. par div. Savans* **6**:621-56.

Lee ET (1992) *Statistical Methods for Survival Data Analysis*. New York: John Wiley and Sons.

Levy P (1939) *Composita Mathematica* **7**:283-339.

Maxwell JC (1860a) *Phil Mag* **19**:19

Maxwell JC (1860b) *Phil Mag* **20**:21, 33.

Metropolis N, Rosenbluth A, Rosenbluth M, Teller A, and Teller E (1953) Equation of state calculations by fast computing machines. *J. Chem. Phys.* **21**:1087-90.

Nelson W (1982) *Applied Life Data Analysis*. New York: John Wiley and Sons.

Nelder JA, and Mead R (1965) A simplex method for function minimization. *Computer Journal* **7**:308-13.

Pearson K (1895) *Phil. Trans. Roy. Soc. London, Series A* **186**:343-414.

Pearson K (1900) *Phil Mag and J Sci*, 5th Series. **50**:157-75.

Pickles A (1985) *An Introduction to Likelihood Analysis.* Norwich: Geobooks.

Powell MJD (1964) An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Comp. Journal* **7**:155-62.

Press WH, Flannery BP, Teukolsky SA, Vetterling WT (1989) *Numerical Recipes in Pascal: The Art of Scientific Programming*. Cambridge: Cambridge University Press.

Rao CR (1973) *Linear Statistical Inference and Its Applications*. New York: John Wiley and Sons.

Ridders CJF (1982) Advances in Engineering Software **4**(2):75-6.

SAS Institute (1985) *SAS User's Guide: Statistics*. Version 5 edition. Cary, NC: SAS Institute, Inc.

Schrödinger E (1915) *Phys. Zeit.* **16**:289-95.

Subbotin MT (1923) *Mathematicheskii Sbornik* **31**:296-301.

Thomas M (1949) *Biometrika* **36**:18-25.

Tuma NB, Hannan MT (1984) *Social Dynamics: Models and Methods*. New York: Academic Press.

Tweedie MCK (1947) *Proc. Camb. Phil. Soc.* **43**:41-9

Vaupel JW (1990) Relatives' risks: Frailty models of life history data. *Theor Pop Biol* **37**:220-34.

Vaupel JW, Yashin AI (1985) Heterogeneity's ruses: Some surprising effects of selection on population dynamics. *Am Stat* **39**:176-85.

Wald A (1947) *Sequential Analysis* New York:John Wiley & Sons.

Wise ME (1966) *Acta Phys. Pharm. Neerlandica* **14**:175-204.

Wood JW (1989) Fecundity and natural fertility in humans. *Oxf Rev Reprod Biol* **11**:61-109.

Wood JW (1994) *Dynamics of Human Reproduction: Biology, Biometry, Demography*. Hawthorne, NY: Aldine de Gruyter.

Wood JW, Holman DJ, Yasin A, Peterson RJ, Weinstein M, Chang M-c (1994) A multistate model of fecundability and sterility. *Demography* **31**:403-26.

Wood JW, Holman DJ, Weiss KM, Buchanan AV, LeFor B (1992) Hazards models for human biology. *Yrbk Phys Anthropol* **35**:43-87.